王道考研操作系统知识点整理

wizardforcel

Published
with GitBook



目錄

介紹 0

第一章 操作系统概述 1

1.1、操作系统的概念、特征、功能和结构 1.1

1.2、操作系统的发展与分类 1.2

1.3、操作系统的运行环境 1.3

1.4、本章疑难点 1.4

第二章 进程管理 2

2.1、进程与线程 2.1

2.2、线程的调度 2.2

2.3、进程同步 2.3

2.4、死锁 2.4

2.5、本章疑难点 2.5

第三章 内存管理 3

3.1、内存管理基础 3.1

3.2、虚拟内存管理 3.2

3.3、本章疑难点 3.3

第四章 文件管理 4

4.1、文件系统基础 4.1

4.2、文件系统实现 4.2

4.3、磁盘组织与管理 4.3

4.4、本章疑难点 4.4

第五章 输入输出管理 5

5.1、IO管理概述 5.1

5.2、IO核心子系统 5.2

5.3、本章疑难点 5.3

王道考研 操作系统知识点整理

第一章 操作系统概述

1.1、操作系统的概念、特征、功能和结构

1、操作系统的概念

在信息化时代，软件被称为计算机系统的灵魂。而作为软件核心的操作系统，已经与现代计算机系统密不可分、融为一体。计算机系统自下而上可粗分为四个部分：硬件、操作系统、应用程序和用户。操作系统管理各种计算机硬件，为应用程序提供基础，并充当计算机硬件和用户的中介。

硬件，如中央处理器、内存、输入输出设备等，提供了基本的计算资源。应用程序，如字处理程序、电子制表软件、编译器、网络浏览器等，规定了按何种方式使用这些资源来解决用户的计算问题。操作系统控制和协调各用户的应用程序对硬件的使用。

综上所述，操作系统是指控制和管理整个计算机系统的硬件和软件资源，并合理的组织调度计算机的工作和资源的分配，以提供给用户和其他软件方便的接口和环境集合。计算机操作系统是随着计算机研究和应用的发展逐步形成并发展起来的，它是计算机系统中最基本的系统软件。

2、操作系统的特征

操作系统是一种系统软件，但与其他的系统软件和应用软件有很大的不同，他有自己的特殊性即基本特征，操作系统的基本特征包括并发、共享、虚拟和异步。这些概念对理解和掌握操作系统的核心至关重要，将一直贯穿于各章节中。

（1）并发

并发是指两个或多个事件在同一时间间隔内发生，在多道程序环境下，一段时间内宏观上有多个程序在同时执行，而在同一时刻，单处理器环境下实际上只有一个程序在执行，故微观上这些程序还是在分时的交替进行。操作系统的并发是通过分时得以实现的。操作系统的并发性是指计算机系统中同时存在多个运行着的程序，因此它具有处理和调度多个程序同时执行的能力。在操作系统中，引入进程的目的实施程序能并发执行。

（2）共享

资源共享即共享，是指系统中的资源可供内存中多个并发执行的进程共同使用。共享可以分为以下两种资源共享方式。

1）互斥共享方式

系统中的某些资源，，如打印机、磁带机，虽然他们可以提供给多个进程使用，但为使所打印的内容不致造成混淆，应规定在同一段时间内只允许一个进程方位该资源。

为此，当进程a访问某资源时，必须先提出请求，如果此时该资源空闲，系统便可将之分配给进程a使用，伺候若再有其他进程也要访问该资源（只要a未用完）则必须等待。仅当进程a访问完并释放该资源后，才允许另一进城对该资源进行访问。计算机系统中的大所属物理设备，以及某些软件中所用的栈、变量和表格，都属于临界资源，他们都要求被互斥的共享。

2）同时访问方式

系统中还有一种资源，允许在一段时间内由多个进程“同时”对它进行访问。这里所谓的“同时”往往是宏观上的，而在微观上，这些进程可能是交替的对该资源进行访问即“分时共享”。典型的可供多个进程同时访问的资源是磁盘设备，一些用重入码编写的文件也可以被“同时”共享，即若干个用户同时访问该文件。

并发和共享是操作系统两个最基本的特征，这两者之间又是互为存在条件的：1资源共享是以程序的并发为条件的，若系统不允许程序并发执行，则自然不存在资源共享的问题；2若系统不能对资源共享实施有效地管理，也必将影响到程序的并发执行，甚至根本无法并发执行。

（3） 虚拟

虚拟是指把一个物理上的实体变为若干个逻辑上的对应物。物理实体是实的，即实际存在的；而后者是虚的，是用户感觉上的事物。相应的，用于实现虚拟的技术，成为虚拟技术。在操作系统中利用了多种虚拟技术，分别用来实现虚拟处理器、虚拟内存和虚拟外部设备。

在虚拟处理器技术中，是通过多道程序设计技术，让多道程序并发执行的方法，来分时使用一台处理器的。此时，虽然只有一台处理器，但他能同时为多个用户服务，是每个终端用户都认为是有一个中央处理器在为他服务。利用多道程序设计技术，把一台物理上的CPU虚拟为多台逻辑上的CPU，称为虚拟处理器。

类似的，可以通过虚拟存储器技术，将一台机器的物理存储器变为虚拟存储器，一边从逻辑上来扩充存储器的容量。当然， 这是用户所感觉到的内存容量是虚的，我们把用户所发哦绝倒的存储器程序虚拟存储器。

还可以通过虚拟设备技术，将一台物理IO设备虚拟为多台逻辑上的IO设备，并允许每个用户占用一台逻辑上的IO设备，这样便可使原来仅允许在一段时间内有一个用户访问的设备，变为在一段时间内允许多个用户同时访问的共享设备。

因此操作系统的虚拟技术可归纳为：时分复用技术和空分复用技术。

（4） 异步

在多道程序环境下，允许多个程序并发执行，但由于资源有限，进程的执行不是一贯到底，而是走走停停，以不可预知的速度向前推进，这就是进程的异步性。

异步性使得操作系统运行在一种随机的环境下，可能导致进程产生于时间有关的错误。但是只要运行环境相同，操作系统必须保证多次运行进程，都获得相同的结果。

3、操作系统的目标和功能

为了给多道程序提供良好的运行环境，操作系统应具有几方面的功能：处理器管理、存储器管理、设备管理和文件管理。为了方便用户使用操作系统，还必须向用户提供接口。同时操作系统可用来扩充机器，以提供更方便的服务、更高的资源利用率。

（1）操作系统作为计算机系统资源的管理者

1）处理器管理

在多道程序环境下，处理器的分配和运行都是以进程为基本单位，因而对处理器的管理可归结为对进程的管理。进程管理的主要功能有：进程控制，进程同步，进程通信，死锁处理，处理器调度等。

2）存储器管理

存储器管理的主要任务是位多通道程序的运行提供良好的环境，方便用户使用以及提高内存的利用率。因此，存储器管理应具备：内存分配、地址映射、内存保护与共享和内存扩充等。

3）文件管理

文件管理主要包括文件的存储空间管理、目录管理及文件读写管理及保护等。

4）设备管理

设备管理的主要任务就是完成用户的IO请求，方便用户使用各种设备，并提高设备的利用率，主要包括混充管理、设备分配、设备处理和虚拟设备等功能。

（2）操作系统作为用户与计算机硬件系统之间的接口

为方便用户使用操作系统，操作系统还提供了用户接口。操作系统提供的接口主要分为两类：一类是命令接口，用户利用这些操作命令来组织和控制作业的执行；另一类是程序接口，编程人员可以使用它们来请求操作系统服务。

1）命令接口

使用命令接口进行作业控制的主要方式有两种：按作业控制方式的不同，可以将命令接口分为联机命令接口和脱机命令接口。

2）程序接口

程序接口由一组系统调用命令组成。用户通过在程序中使用这些系统调用命令拉i请求操作系统提供的服务。用户在程序中可以直接使用这组系统调用命令向系统提出各种服务请求，如使用各种外部设备，进行有关磁盘文件的操作，申请分配和收回内存 以及其他各种控制要求。

所谓系统调用就是用户在程序中调用操作系统所提供的一些子功能。具体的讲，系统调用就是通过系统调用命令中断现行程序，而转去执行响应的子程序，以完成特定的系统功能；系统调用完成后，返回程序的断点以继续执行。

系统调用命令是作为扩充机器命令提供的，目的是增强系统功能，方便用户使用。而起通过系统调用的方式来使用系统功能，可以保证系统的稳定性和安全性，防止用户随意更改或访问系统的数据或命令。因此，在一些计算机系统中，把系统调用命令成为广义指令。广义指令与机器指令在性质上是不同的，机器指令使用硬件电路直接实现的，而广义命令则是由操作系统提供的一个或多个子程序模块实现的。显然，系统调用属于核心态指令。

没有任何软件支持的计算机成为裸机，它仅构成计算机系统的物质基础，而实际呈现在用户面前的计算机系统是经过若干层软件改造的计算机。裸机在最里层，他的外面是操作系统，有操作系统提供的资源管理功能和方便用户的各种服务功能，将裸机改造成功能更强、使用更方便的机器，通常把覆盖了软件的机器成为扩充机器，又称之为虚拟机。

4操作系统的结构

像现在操作系统这样庞大而复杂的系统，为了能正常工作且容易修改和维护，在实现前必须认真设计操作系统的结构。操作系统发展至今，其设计结构可以分成以下几类：

（1） 简单结构。

（2） 模块化结构

（3） 分层式结构

（4） 微内核结构

1.2、操作系统的发展与分类

1、手工操作阶段

2、脱机输入输出阶段

3、批处理阶段

批处理技术是指计算机系统对一批作业自动进行处理的一种技术。批处理阶段的特点是：用户不用与计算机直接打交道，而是通过专门的操作员来完成作业的输入输出。随着外围设备的迅速发展，后来又出现了脱机批处理系统，即主机直接与磁盘通信。

（1） 单道批处理系统

主要特点：自动性、顺序性、单道性。

（2） 多道批处理系统

多道程序设计技术是指在计算机内存中同时存放几道相互独立的程序，它们在管理程序的控制下相互交替的运行。其特征是：多道，宏观上并行，微观上串行。

4、分时操作系统

所谓分时系统就是把处理器的运行时间分成很短的时间片，按时间片轮流把处理器分配给各联机作业使用。若某个作业再分配给他的时间片内不能完成其计算，则改作业暂时停止运行，把处理器让给其他作业使用，等待下一轮再继续运行，由于计算机速度很快，作业运行轮转的很快，给每个用户的感觉好像是自己独占一台计算机。

分时操作系统十多个用户通过终端同事共享一台主机，这些终端连接在主机上，用户可以同时与主机进行交互操作而不互相干扰。所以，实现分时系统最关键的问题，是如何使用户能与自己的作业进行交互，即当用户在自己的中断上输入命令时，系统应能及时接收并及时处理该命令，再将结果返回用户。分时系统也是支持多道程序设计的系统，但它不同于多道批处理系统。多道批处理是实现作业自动控制而无需人工干预的系统，而分时系统是实现人机交互的系统，这使得分时系统具有与批处理系统不同的特征，其主要特征如下：

同时性，交互性，独立性，及时性。

5、实时操作系统

实时系统的主要特点是：实时性和可靠性。

6、网络操作系统和分布式计算机系统

7、个人计算机操作系统

1.3、操作系统的运行环境

1、操作系统的运行机制

计算机系统中，通常CPU执行两种不同性质的程序，一种是操作系统内核程序；另一种是用户自编程序或系统外城的应用程序。对操作系统而言，这两种程序的作用不同，前者是后者的管理者和控制者，因此“管理程序”要执行一些特权指令，而“被管理程序”出于安全性考虑，不能执行这些指令。所谓特权指令，是指计算集中不允许用户直接使用的指令，如IO指令、置中断指令。

操作系统在具体实现上划分了用户态和核心态，以严格区分两种类程序。

一些与硬件关联交紧密的模块，诸如时钟管理程序、中断处理程序、设备驱动程序等处于最底层。其次是运行频率较高的程序，诸如金城关里、存储器管理和设备管理等。这两部分内容构成了操作系统的内核。这部分内容的指令操作工作在核心态。

内核是计算机上配置的最底层软件，是计算机功能的眼神。不同系统对内核的定义稍有区别，大多数操作系统内核包括四个方面的内容。

（1） 时钟管理

在计算机外部设备中，时钟是最关键的设备。时钟的第一功能是计时，操作系统需要通过时钟管理，向用户提供标准的系统时间。另外，通过时钟中断的管理，可以实现京城的切换。诸如：在分时操作系统中，采用时间片轮转调度的实现；在实时系统中，按截止时间控制运行的实现；在批处理系统中，通过时钟管理来衡量一个作业的运行程度等。因此，系统管理的方方面面无不依赖于它。

（2） 中断机制

引入中断技术的初衷是提高多道程序运行环境中CPU的利用率，而且主要是针对外部设备的。后来的到发展，形成了多种类等，成为操作系统各项操作的基础。例如键盘或鼠标信息的输入、进程的管理和调度、系统功能的调用、设备驱动、文件访问等，无不依赖于中断机制。可以说，现代计算机系统是靠中断驱动的软件。

（3） 原语

按层次结构涉及的操作系统，底层必然是一些可被调用的公用小程序，他们各自完成一个规定的操作。其特点是：1.他们处于操作系统的最底层，是最接近硬件的部分。2.这些程序的运行具有原子性——其操作只能一起合成。3.这些程序的运行时间都较短，而且调用频繁。

通常把具有这些特点的程序成为原语。定义源于的直接方法是关闭中断，让她的所有动作不可分割的进行完再打开中断。

系统中的设备驱动、CPU切换、进程通信等功能中的部分操作都可以定义为原语，是他们呢成为内核的组成部分。

（**4**） 系统控制的数据结构及处理

系统中用来登记状态信息的数据结构很多。比如作业控制块、进程控制块、设备控制块、各类链表、消息队列、缓冲区、空闲区登记表、内存分配表等。为了实现有效地管理 ，系统需要一些基本的操作，常见的操作有以下三种：

1） 进程管理：进程状态管理、进程调度和分配、创建与撤掉进程控制块的队列维护操作等。

2） 存储器管理：存储器的空间分配和回收管理、内存信息保护程序、代码对换程序等。

3） 设备管理：缓冲区管理、设备分配和回收等。

从上述内容可以了解，核心态指令实际上包括系统调用类指令和一些针对时钟、中断和原语的操作指令。

2、中断和异常的概念

在操作系统中引入核心态和用户态这两汇总工作状态后，就需要考虑这两种状态之间如何切换。操作系统内核工作在核心态，而用户程序工作在用户态。但系统不允许用户程序实现核心态的功能，而它们又必须使用这些功能。

中断(interruption)也成为外中断，指来自CPU执行指令以外的事件发生，如设备发出的IO结束中断，表示设备输入输出处理已经完成，希望处理器能够像设备发出下一个输入输出请求，同事让王成输入输出后的程序继续进行。时钟中断，表示一个固定的事件篇已到，让处理器处理计时、启动定时运行的任务。这一类中断通常是与当前运行的程序无关。中断可细分为硬中断和软中断。硬中断是硬件产生的；软中断是软件产生的。

异常(Exception)也成为内中断、例外或陷入。指源自CPU执行指令内部的时间，如程序的非法操作码、地址越界、算数一出、虚存系统的缺页以及专门的陷入指令等引起的时间。对异常的处理一般要依赖与当前程序的运行现场，而且异常不能被屏蔽，一旦出现异常立即处理，而关于内中断和外中断的联系与区别如下：

这样，操作系统的运行环境可以理解为：用户通过操作系统运行上层程序，而这个上层程序的运行以来与操作系统的底层管理程序提供服务支持，当需要管理程序服务时，系统则通过硬件中断机制进入核心态，运行管理程序；另外也可能是程序运行出现异常情况，被动的需要管理程序的服务，这是就通过异常处理来进入核心态。当管理程序运行结束时，用户程序要继续进行，则通过相应保存程序现场推出中断处理程序或异常处理程序，返回断点处继续执行。在操作系统这一层面上，我们关心的是系统核心态和用户态的软件实现和切换，对于硬件层面的具体理解，可以结合“计算机组成原理”课程中有关中断的内容进行学习。

下面列举一些由用户态转向核心态的例子：

1） 用户程序要求操作系统的服务，即系统服务。

2） 发生一次中断

3） 用户程序中产生了一个错误状态

4） 用户程序中企图执行一条特权指令

5） 从核心态转向用户态由一条指令实现，这条指令也是特权指令。一般情况是中断返回指令。

注意，由用户态进入核心态，不仅仅是状态需要切换，而且，所使用的对战也可能需要由用户堆栈切换为系统对战，但这个系统对战也是属于该进程的。

1.4、本章疑难点

1.并行性与并发性的区别和联系

并行性和并发性是既相似又有区别的两个概念。并行性是指两个或多个事件在同一时刻发生，并发性是指两个或多个事件在同一时间间隔内发生。

2.分时系统与实时系统特征的比较

3.特权指令的工作机制

所谓特权指令是指有特殊权限的指令，由于这类指令的全县最大，如果使用不当，就会皮怀系统或其他用户信息。为了保证系统安全，这类指令只能用于操作系统或其他系统软件，不直接提供给用户使用，主要用于系统资源的分配和管理，包括改变系统的工作方式，检测用户的访问权限，修改虚拟存储器管理的段表、页表和完成人五的创建和切换等。在某些用户的计算机系统中，为了统一管理各种外部设备，输入输出指令也作为特权指令，不允许用户直接使用。需要输入输出操作时，必须通过系统调用，经由操作系统完成。特权指令那个必须在核心态之星，核心态又叫做特权态、系统态。实际上，CPU在核心态的下可以执行指令系统的全集。

为了防止用户系统中使用特权指令，用户态下只能使用除特权指令以外的指令，核心态下可以使用全部指令。所以把用户程序放在用户态下进行，而操作系统中必须使用特权指令的那部分程序在核心态下运行，保证了计算机系统的安全可靠。从用户态转换为核心态的唯一途径就是终端或异常。

4.系统调用产生的访管中断

程序员在编写程序时，因要求操作系统提供服务而有意识的使用“访管指令”，从而导致程序中断，这属于一种自愿性的中断，所以又称其为“访管中断”。“访管”中断是由访管指令产生，程序员可以使用访管指令中设置参数，当CPU执行到访管指令那个时，将访管指令中的操作数存入到主存中的约定单元，然后产生访管中断，引出操作系统来处理访管中断中的具体要求。这种利用访管指令来实现的指令成为广义指令。当初与用户态的用户程序使用访管指令时，系统根据该访管指令的操作数执行访管中断处理程序，访管中断处理程序将按系统调用的操作数和参数转到响应的例行子程序。完成服务功能后，退出中断，返回到用户程序断点继续执行。

第二章 进程管理

进程管理是操作系统的核心内容，也是每年必考的重点。其中，进程概念、进程调度、信号量机制实现同步和互斥、进程死锁等更是重中之重，要求熟练掌握。此外，需要注意的是：本章除了选择题外还很容易考察综合题，在最近三年的考研综合题有两道题考察了本章的知识点。其中信号量实现进程同步和互斥，进程调度算法和银行家算法都是可能出现的综合题考点，需要读者多加注意。

2.1、进程与线程

1、进程的概念和特征

（1） 进程的概念

在多道程序环境下，允许多个程序并发执行，此时他们将失去封闭性，并具有间断性和不可再现性的特征。为此引入了进程的概念，以便更好地描述和控制程序的并发执行，实现操作系统的并发行和共享性。为此引入了进程的概念，以便更好地描述和控制程序的并发执行，实现操作系统的并发性和共享性。

为了是参与并发执行的程序能独立的运行，必须为之配置一个专门的数据结构，称之为进程控制块（process control block），系统利用PCB来描述进程的基本情况和运行状态，进而控制和管理进程。

相应的，有程序段、相关数据段和PCB三部分构成了进程映像（进程实体）。所谓创建进程，实质上是创建进程映像中的PCB；而撤销进程，实质上是撤销进程的PCB。指的注意的是，进程影响是静态的，晋城市动态的。

从不同的角度，进程可以有不同的定义，比较经典的定义有：

1） 进程是程序的一次执行过程

2） 进程是一个程序及其数据在处理器上顺序执行时所发生的活动。

3） 进程是具有独立功能的程序在一个数据集合上运行的过程，他是系统进行资源分配和调度的一个独立单位。

在引入了进程实体的概念后，我们可以吧传统的操作系统中的进程定义为："进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位"。

（2） 进程的特征

进程是由多程序的并发执行而引出的，他和程序是两个截然不同的概念。进程的基本特征是对比单个程序的顺序执行提出的，也是对进程管理提出的基本要求。

1） 动态性：进程是程序的一次执行，他有着创建、活动、暂停、终止等过程，具有一定的生命周奇奇，是动态的产生、变化和消亡的。动态性是进程最基本的特征。

2） 并发性：至多个进程实体，同存于内存中，能在一段时间内同时运行，并发性是进程的重要特征，同时也是操作系统的重要特征，引入进程的目的就是为了是程序能与去其他进程的程序并发执行，以提高资源利用率。

3） 独立性：指进程实体是一个能独立运行、独立获得资源和独立接收调度的基本单位。范围建立PCB的程序都不能作为一个独立的单位参与运行。

4） 异步性：由于进程的相互制约，是进程具有执行的间断性。也即进程按各自独立的、不可预知的速度向前推进。异步性会导致执行结果不可再现性，为此，在操作系统中必须配置相应的进程同步机制。

5） 结构性：每个进程都配置一个PCB对其进行描述。从结构上来看，进程实体是由程序段、数据段和进程控制端三部分组成的。

2、进程的状态与转换

进程在其生命周期内，由于系统中个进程之间的相互制约关系以及系统的运行环境的变化，使的进程的状态也在不断地发生着变化。通常进程有以下五种状态。前三种是进程的基本状态。

1） 运行状态：进程正在处理器上运行。在单处理器的环境下，每一时刻最多只有一个进程处于运行状态。

2） 就绪状态：进程已处于准备运行的状态，即进程获得了除CPU之外的一切所需资源，一旦得到处理器即可运行。

3） 阻塞状态：又称为等待状态：进程正在等待某一事件而暂停运行，如等待某资源为可用（不包括处理器），或等待输入输出的完成。及时处理器空闲，该进程也不能运行。

4） 创建状态：进程正在被创建，尚未转到就绪状态。创建进程通常需要多个步骤：首先申请一个空白的PCB，并向PCB中填写一些控制和管理进程的信息；然后由系统为该进程分配运行时所必须的资源；最后把该进程转入到就绪状态。

5） 结束状态：进程正在从系统中消失，这可能是进程正常结束或其他原因中断退出运行。当进程需要结束运行时，系统首先必须置该进程为结束状态，然后再进一步处理资源释放和回收工作。

注意区别就绪状态和等待状态：就绪状态是指进程仅缺少处理器，只要活得处理器资源就立即执行；而等待状态是指进程需要其他资源或等待某一事件，及时处理器空闲也不能运行。

3、进程控制

进程控制的主要功能是对系统中所有进程实施有效地管理，她具有创建新进程、撤销已有进程、实现进程状态转换等功能。在操作系统中，一般把进程控制用的程序段成为原语，原语的特点是执行期间不允许中断，他是一个不可分割的基本单位。

允许一个进程创建另一个进程。

操作系统创建一个新进程的过程如下（创建原语）：

1） 为新进程分配一个为我一个进程标示号，并申请一个空白的PCB。

2） 为进程分配资源，为新进程的程序和数据，以及用户占分配必要的空间。

3）初始化PCB，主要包括初始化标识信息、初始化处理器状态信息和初始化处理器控制信息，以及设置进程的空闲及。

4）如果进程就绪队列能够接纳新进程，就将新进程插入到就绪队列，等待被调度运行。

引起进程终止的时间主要有：正常结束、表示进程的任务已经完成和准备退出运行。异常结束是指进程在运行时，发生了某种异常事件，是程序无法继续运行，如：存储区越界、保护措、非法指令、特权指令错、IO故障等。外界干预是指进程外界的请求而终止，如操作员或操作系统干预、父进程请求和父进程终止。

操作系统终止进程的过程如下：（撤消原语）

1）根据被终止进程的标示符，检索PCB，从中读出该进程的状态。

2）若被终止进程处于执行状态，立即终止该进程的执行，将处理器资源分配给其他进程。

3）若该进程还有子进程，则应将其所有子进程终止。

4）将该进程所拥有的资源、或归还给父进程或归还给操作系统。

5）将该PCB从所在队列（链表）中删除。

（**3**）进程的阻塞和唤醒

正在执行的进程，犹豫期待的某些时间为发生，如请求系统资源失败、等待某种操作的完成、新数据尚未到达或无心工作可做等，则由系统自动执行阻塞原语，使自己由运行状态变为阻塞状态。可见，进程的阻塞是进程自身的一种主动行为。

阻塞原语的执行过程为：找到将要被阻射进城的标识号对应的PCB，如果该进程为运行状态，则保护其现场，将其状态改为阻塞状态，停止运行，并把该PCB插入响应时间的等待队列中去；若为就绪状态，则将其状态改为阻塞状态，把它溢出就绪队列，插入到等待队列中去。

当阻塞进程所期待的时间出现时，如它所启动的IO操作已完成或其所期待的数据已到达，则有关进程（比如，提供数据的进程），调用唤醒原语，将等待该事件的进程唤醒，唤醒原语的执行过程是：在该事件的等待队列中找到相应进程的PCB，然后把该PCB插入到就绪队列中，等待调度程序调度。

需要注意的是，Block原语和Wakeup原语是一对作用刚好相反的原语，必须成对使用。Block原语是由被阻塞进程自我调用实现的，而Wakeup原语则是由一个与被唤醒进程相合作或被其他相关进程调用实现的。

无论什么样的进程操作，都是在内核执行的。

进程切换是指当前正在运行的进程被转换到其他状态后，再回到运行继续执行的过程，这个过程中，进程的运行环境产生了实质性的变化。进程切换的过程如下：

1）保存处理器上下文，包括程序计数器和其他寄存器。

2） 更新PCB信息。

3） 把进程的PCB移入相应的队列，如就绪、在某时间阻塞等队列。

4） 选择另一个进程执行，并更新其PCB。更新内存管理的数据结构。

5） 恢复处理器的上下文。

4、进程的组织

进程是操作系统的资源分配和独立运行的基本单位。

（1） 进程控制块

进程创建时，操作系统就新建一个PCB结构，它之后就常驻内存，任意时刻可以存取。在进程结束时删除。PCB是进程实体的一部分，是进程存在的唯一标识。

PCB主要包括：进程描述信息、进程控制和管理信息、资源分配清单和处理器相关信息等。

在一个系统中，通常存在这许多进程，有的处于就绪状态，有的处于阻塞状态，而且阻塞的原因各不相同。为了方便进程的调度和管理，需要将各进程的PCB用适当的方法组织起来。目前，常用的组织方式有连接方式和索引方式两种。连接方式将同一状态的PCB连接成一个队列，不同状态对应不同的队列，也可以把处于阻塞状态的进程的PCB，根据其阻塞原因的不同，排成多个阻塞队列。索引方式是将同一状态的进程组织在一个索引表中，索引表的表项只想相应的PCB，不同状态对应不同的索引表，如就绪索引表和阻塞索引表等。

程序段就是能北京城调度程度调度到CPU执行的程序代码段。注意，程序可以被多个进程共享，就是说多个进程可以运行同一个程序。

一个进程的数据段，可以是进程对应的程序加工处理的原始数据，也可以是程序执行时产生的中间或最终结果。

5、进程的通信

进程通信就是进程之间的数据交换。PV操作时低级通信方式2，高级通信方式是指以较高的效率传输大量数据的通信方式。高级通信方法可分为共享存储、消息传递和管道通信三大类。

（1） 共享存储

在通信的进程之间存在着一款可以直接访问的共享空见，通过对这块共享空间的读写操作时间进程之间的信息交换。在共享存储方法中，需要使用同步互斥工具。

需要注意的是：用户进程空间一般都是相互独立的，要想让两个用户进程共享空间，必须通过特殊系统调用实现，而进程内的线程是自然共享进程空间的。

（2） 消息传递

在消息传递系统中，进程间的数据交换，是以格式化的小心Message为单位的。

（3） 管道通信

管道通信是消息传递的一种特殊方式。。所谓管道，就是用于连接一个读进程和一个写进程以实现他们之间通信的一个共享文件，又名为pipe文件。向管道或共享文件提供输入的发送进程，以字符流的形势将大量的数据送入写管道；而接收管道输出的接收进城，则从管道中接受数据。为了协调双方的通信，关到极致必须他提供以下撤按方面的协调能力：互斥、同步和确定对方存在。

6、线程概念和多线程模型

引入进程的目的，是为了是多道程序能并发执行，以提高资源利用率和系统吞吐量；而引入线程，则是为了减小程序在并发执行时所付出的时空开销，提高操作系统的并发性能。

线程最直接的理解就是“轻量级进程”，它是一个基本的CPU执行单元，也是程序执行流的最小单元，由线程ID、程序计数器、寄存器集合和堆栈组成。线程是进程中的一个实体，是被系统独立调度和分派的基本单位。进程只作为除CPU以外的系统资源的分配单元，线程则作为处理器的分配单元。线程也有就绪、阻塞和运行三种基本状态。

（2） 线程和进程的比较

1） 调度：在引入线程的操作系统中，线程是独立调度的基本单位，进程是资源拥有的基本单位。

2） 拥有资源：进程是拥有资源的基本单位，而线程不拥有系统资源，单线程可以防伪其隶属进程的系统资源。

3） 并发性：在引入线程的操作系统中，不仅进程之间可以并发执行，线程之间也可以并发执行，从而是操作系统具有更好的并发性，大大提高了系统的吞吐量。

4） 系统开销：线程开销极小。

5） 地址空间和其他资源：进程的地址空间之间相互独立，同一进程的各线程间共享进程的资源，进程内的线程对进程外的其他进程不可见。

6） 通信方面：进程间通信需要进程同步和互斥手段的辅助，以保证数据的一致性，而线程间可以直接读写进程数据段来进行通信。

（3） 线程的属性

在多线程操作系统中，八仙城作为独立运行的基本单位。此时的进程已不是一个基本可执行的实体。线程的主要属性如下：

1） 线程是一个轻型实体，它不拥有系统资源，但每个线程都应有一个唯一的标识符和一个线程控制块，线程控制块记录了线程执行的寄存器和栈等现场情况。

2）不同的线程可以执行相同的程序，即同一个服务程序被不同的用户调用时，操作系统为他们创建不同的线程。

3）统一进程中的各个线程共享该进程所拥有的系统资源。

4）线城市处理器的独立调度单位，多个线程是可以并发执行的。

5）一个县城被创建后便开始了它的生命周期，直至终止，线程在生命周期内会经历等待态、就绪态和运行态等各种状态变化。

（4）线程的实现方法

线程的实现可以分为两类：用户级线程和内核级线程。

（5）多线程模型

有些系统同时支持用户线程和内核线程，由此产生了不同的多线程模型，即实现用户级线程和内核级线程的连接方式。

1）多对一模型。多对一模型将多个用户级线程映射到一个内核级线程。线程管理在用户空间完成。

2）一对一模型。

3）多对多模型。

特点：克服了多对一模型的并发度不高的缺点，又克服了一对一模型中一个用户进程占用太多内核级线程，开销太大的缺点。

2.2、线程的调度

1、调度的概念

在多道程序系统中，进程的数量往往多于处理器的个数，进程争用处理器的情况在所难免。处理器调度是对处理器进行分配，就是从就绪队列中，按照一定的算法，选择一个进程并将处理器分配给他运行，以实现进程的并发执行。

处理器调度是多道程序操作系统的基础，它是操作系统设计的核心问题。

一个作业从提交开始知道完成，往往要经历一下三级调度：

1）作业调度。作业调度又称高级调度：其主要任务是按一定的原则从外存上处于后备状态的作业中挑选一个或多个作业，给他们分配内存、输入输出设备等必要的资源。并建立相应的进程，以使他们获得竞争处理器的权利。

多道批处理系统中大多配有作业调度，而其它系统中通常不需要配置作业调度。作业调度的执行频率较低，通常为几分钟一次。

2）中级调度。中级调度又称内存调度。引入中级调度视为了提高内存利用率和系统吞吐率，为此，应使那些暂时不能运行的进程调至外存等待，把此时的进程状态称为挂起状态。当他们已具备运行条件且内存有稍有空闲时，由中级调度来决定，吧外存上那些已具备运行条件的就绪进程，在重新调入内存，并修改其状态为就绪状态，挂在就绪队列上等待。

3）进程调度。进程调度又称为低级调度，其主要任务是按照某种方法和策略从就绪队列中选取一个进程，将处理器分配给它。进程调度是操作系统中最基本的一中调度，在一般操作系统中都不需配置进程调度。进程调度的频率很高，一般几十毫秒一次。

作业调度从外存的后备队列中选择一批作业进入内存，为他们建立进程。这些进程被送入就绪队列。进程调度从就绪队列中选出一个进程，并把其状态改为运行状态，把CPU分配给它。中级调度是位于高级调度和低级调度之间的一种调度。为了提高内存的利用率，系统将那些暂时不能运行的进程挂起来。当内存空间宽松式，通过中级调度选择具备运行条件的进程，将其唤醒。

2、调度的时机、切换与过程

进程调度和切换程序是操作系统内核程序。当请求调度的事件发生后，才可能会运行进程调度程序，当调度了新的就绪进程后，才会去进行进程间的切换。

现在操作系统中，不能进行进程的调度与切换的情况有以下几种：

1） 在处理中断的过程中：中断处理过程复杂，在实现上很难做到，而且中断处理时系统工作的一部分，逻辑上不属于某一进城，不应被剥夺处理器资源。

2） 进程在操作系统内核程序临界区中：进入临界区后，需要独占式的访问共享数据，理论上必须加锁，以防止其他并行程序的进入，在解锁前不应该切换到其他进程，以加快该共享数据的释放。

3） 其他需要完全屏蔽中断的原子操作过程中：如加锁、解锁、中断现场保护、恢复等等源自操作。在原子过程中，连中断都要屏蔽，更不应该进行进程的切换。

如果在上述过程中发生了引起调度的条件，并不能马上进行调度和切换，应置系统请求调度标志，知道上述过程结束后才能进行相应的调度和切换。

应该进行进程的调度与切换的情况有：

1） 当发生引起调度条件且当前进程无法继续运行下去时，可以马上进行调度与切换。如果操作系统只在这种情况下进行进程调度，就是非剥夺调度。

2） 当中断处理结束后或自陷处理结束后，返回被中断进程的用户态程序执行现场前，若置上请求调度标志，即可马上进行进程调度与切换。如果操作系统支持这种情况下的运行调度程序，就实现了剥夺方式的调度。

进程切换往往在调度完成后立刻发生，它要求保存源进程当前切换点的县城信息，恢复被调度进程的现场信息。现场切换时，操作系统内核将远近程的现场信息推入到当前进程的内核对战来保存他们，并更新堆栈指针。内核完成从新进程的内核栈中装入新进程的县城信息、更新当前运行进程空间指针、重设PC寄存器等相关工作之后，开始运行新的进程。

3、进程调度方式

所谓进程调度方式是指当某一个进程正在处理器上执行时，若有某个更为重要或紧迫的进程需要处理，既有优先权更高的进程进入就绪队列，此时应如何分配处理器。通常有一下两种进程调度方式：

（1） 非剥夺调度方式

非剥夺调度方式又称为非抢占调度方式，是指当一个进程正在处理器上执行时，即使有某个更为重要或紧迫的进程进入就绪状态，仍然让正在执行的进程继续执行，知道该进程完成或发生某种时间而进入阻塞状态时，才把处理器分配给更为重要或紧迫的进程。

（2） 剥夺调度方式

剥夺调度方式又称为抢占方式，是指当一个进程正在处理器上执行时，若有某个更为重要或紧迫的进程需要使用处理器，则立即暂停正在执行的进程，将处理器分配给这个更为重要或紧迫的进程。

“剥夺”不是一种任意性行为，必须遵循一定的原则：优先权原则，短进程优先原则和时间片原则。

4、调度的基本准则

不同的调度算法具有不同的特性，在选择调度算法时，必须考虑算法所具有的特性。为了比较处理器调度算法的性能，人们提出很多评价准则，下面介绍主要的几种准则：

（1） CPU利用率

CPU是计算机系统中最重要的资源之一，所以应尽可能使CPU保持在忙状态，是这一资源利用率最高。

（2） 系统吞吐量

系统吞吐量表示单位时间内CPU完成作业的数量。长作业需要消耗较长的处理器时间，因此会降低系统的吞吐量。而对于短作业，他们所需要消耗的处理器时间端，因此能提高系统的吞吐量。调度算法和方式的不同，也会对系统的吞吐量产生较大的影响。

（3） 周转时间

周转时间是指从作业提交到作业完成所经历的时间，包括作业等待、在就绪队列中排队、在处理器上运行以及进行输入输出操作所花费的时间的总和。

作业的周转时间=作业完成时间-作业提交时间

（4） 等待时间

等待时间是指进程处于等处理器状态时间之和，等待时间越长，用户满意度越低。处理器调度算法实际上并不影响作业执行或输入输出操作时间，只影响作业在就绪队列中等待所花的时间。因此，衡量一个调度算法优劣常常只需简单地考察等待时间。

（5） 响应时间

响应时间是指从用户提交请求到系统首次产生响应所有的时间。在交互式系统中，周转时间不可能是最好的评测准则，一般采用响应时间作为衡量调度算法的重要准则之一。从用户的角度来看，调度策略应尽量降低响应时间，使响应时间处在用户能够接受的范围之内。

5、典型的调度算法

通常系统的设计目标不同，所采用的调度算法也不同。在操作系统中存在多种调度算法，其中有的调度算法适用于作业调度，有的调度算法适用于进程调度，有的调度算法两者都适用。下面介绍几种常用的调度算法：

（1） FIFS先来先服务调度算法

特点：算法简单，但是效率低；有利于长作业，不利于短作业；有利于CPU繁忙型作业而不利于IO繁忙型作业。

（2） SJF短作业优先调度算法

短作业（进程）优先调度算法是指对短作业祸端进程优先调度的算法。短作业优先调度算法是从后备队列中选择一个或若干个估计运算时间最短的作业，将他们呢掉入内存运行。

SJF调度算法的缺点：

1） 该算法对长作业不理。

2） 该算法完全未考虑作业的紧迫程度

3） 由于作业的长短只根据用户所提供的估计执行时间而定的，而用户又可能会有意或无意的缩短其作业的估计运行时间，致使该算法不一定能真正做到算作业优先调度。

4） 注意：SJF调度算法的平均等待时间、平均周转时间最少。

（3） 优先级调度算法

（4） 高响应比优先调度算法

高响应比优先调度算法主要用于作业调度。同时考虑从每个作业的等待时间和估计需要运行的时间。

（5） 时间片轮转调度算法

时间片轮转调度算法主要适用于分时系统。

（6） 多级反馈队列调度算法

多级反馈队列调度算法主要是时间片轮转调度算法和优先级调度算法的综合和发展。通过动态调整进程优先级和时间片大小，多级反馈队列调度算法可以兼顾多方面的系统目标。

2.3、进程同步

1、进程同步的基本概念

多道程序环境下，进程是并发执行的，不同进程间存在着不同的相互制约关系。为了协调进程之间的相互制约关系，达到资源共享和进程协作，避免进程之间的冲突，引入了进程同步的概念。

（1） 临界资源

多个进程可以共享系统中的各种资源，但其中许多资源一次只能为一个进程所使用，我们把一次只允许一个进程使用的资源成为临界资源。

对临界资源的访问，必须互斥的进行。每个进程中，访问临界资源的那段代码成为临界区。

为了保证临界资源的正确使用，可以把临界资源的访问过程分为四个部分。

1） 进入区。为了进入临界区使用临界资源，在进入去要检查可否进入临界区。

2） 临界区。进程中访问临界资源的那段代码。

3） 退出区。将正在访问临界区的标志清除。

4） 剩余区。代码中的其余部分。

```
do {
entry section;
critical section;
exit section;
remainder section;
}while (true)
```

（2） 同步

同步已成为直接制约关系，它是为完成某种任务而建立的两个或多个进程。这些进程因为需要在某些位置上协调他们的工作次序而等待、传递信息所产生的制约关系。进程间的直接制约关系就是它们之间的相互合作。

（3） 互斥

互斥亦称间接制约关系。当一个进程进入临界区使用临界资源时，另一个进程必须等待，当占用临界资源的进程退出临界区后，另一个进程才允许去访问此临界资源。

2、实现临界区互斥的基本方法

在进入区设置和检查一些标志来表名是否有进程在临界区中，如果已有进程在临界区，则在进入区通过循环检查进行等待，进程离开临界区后则在退出区修改标志。

（3） 硬件实现方法

本节对硬件实现的具体理解对后面的信号量学习很有帮助。计算机提供了特殊的硬件指令，允许对一个字中的内容进行检测和修正，活着是对两个字的内容进行交换等。通过硬件支持实现临界段问题的低级方法或称为元方法。

1） 中断屏蔽方法。当一个进程正在使用处理器执行他的临界区代码时，要防止去其他进程在进入其临界区访问的最简单方法就是禁止一切中断的发生，或称之为屏蔽中断、关中断。因为CPU只有在发生中断时引起进程的调度和切换，这样屏蔽中断就能保证当前运行进程将临界区代码顺利的执行完，然后再开中断。

这种方法限制了处理器交替执行程序的能力，因此执行的效率将会明显降低。对内核来说，当它执行更新变量或列表的几条指令期间关中断是很方便的，但将关中断的权力交给用户则很不明智，若一个进程关中断之后不再打开终端，则系统可能会因此终止。

2） 硬件爱你指令法。

TestAndSet指令：这条指令是原子操作。及执行该代码是不允许被中断。其功能是独处制定标志后把该标志设置为眞。

```
Boolean TestAndSet( Boolean *lock){

Boolean old;

Old=*lock;

*lock=true;

Return old;

}
```

Lock为每个临界资源设置的共享布尔变量，true表示正在被占用，false表示没被占用。

```
While TestAndSet(&lock);

进程的临界区代码；

Lock=false；

进程剩余代码；
```

Swap指令：该指令的功能是交换两个字的内容。

```
Swap(Boolean a, Boolean b){
Boolean temp;
Temp= *a;
a=b;
*b=temp;
}
```

以上对TestAndSet和Swap指令仅仅是功能实现，并非软件实现定义，事实上他们是由硬件逻辑直接实现的，不会被中断。

硬件方法优点：使用与任意数目的进程，不管是单处理器还是多处理器：简单、容易验证其正确性。可以支持进程内有多个临界区，只需要为每个临界区设立一个布尔变量。

硬件方法的缺点：进程等待进入临界区时要耗费处理器时间，不能实现让权等待。从等待进程中随机选择一个进入临界区，有的进程可能一直选不上，从而导致“饥饿”现象。

3、信号量

信号量机构是一种功能较强的机制，可用来解决互斥与同步的问题，它只能被两个标准原语wait和signal来访问，也可以记作p操作和v操作。

原语是指完成某种功能且不被分割不被中断执行的操作序列，有时也成为原子操作，通常可用硬件来实现完成某种功能的不可分割执行特性。

（1） 整形信号量

整形信号量被定义为一个用于表示资源个数的整型量S。

（2） 记录性信号量

记录性信号量是不存在“忙等”现象的进程同步机制。除了需要一个用于代表资源数的整型变量value外，再增加一个进程链表L，用于连接所有等待该资源的进程，记录型信号量是由于采用了记录型的数据结构得名。记录型信号量可描述为：

```
Typedef struct{
Int value；
Struct process * L；
}semaphore；
```

Wait操作，表示进程请求一个该类资源，当S.value<0时，表示该类资源已分配完毕，因此进程调用block原语，进行自我阻塞，放弃处理器，并插入到S.L中，可见该机制遵循了“让权等待”的原则。Signal 操作，表示进程释放一个资源，使系统中可供分配资源数增加一，故S.value++。若加1后仍是S.value<=0，则表示S.L中仍有等待该资源的进程被阻塞，故还应调用wakeup原语，将S.L中的第一个等待进程唤醒。

（3） 利用信号量实现同步

信号量机构能用于解决进程间各种同步问题。

（4） 利用信号量实现互斥

信号量机构能很方便的解决进程互斥问题。

互斥的实现是不同进程对同一信号量进行P操作和V操作，一个进程在成功地对信号量执行了P操作后进入临界区，并在退出临界区后，由该进程本身对该信号量执行V操作，表示当前没有进城进入临界区，可让其他进程进入。

（5） 利用信号量实现前驱关系

信号量也可以用来描述程序之间或者语句之间的前驱关系。

（6） 分析进程同步或互斥问题的方法步骤

1） 关系分析。找出问题中的进程数，并且分析它们之间的同步和互斥关系。同步、互斥、前去关系直接按照上面例子中的经典犯事改写。

2） 整体思路。找出解决问题的关键点，并且根据做过的题目找出解决的思路。根据进程的操作流程确定P操作、V操作的大致顺序。

3） 设置信号量。根据上面两步，设置需要的信号量，确定初值，完善整理。

4、管程

管程是一组数据以及定义在这组数据之上的对这组数据的操作组成的软件模块，这组操作能初始化并改变管程中的数据和同步进程。

1） 局部与管程的共享结构数据说明

2） 对该数据结构进行操作的一组过程

3） 对局部于管程的共享数据设置初始值的语句

1）局部于管程的数据只能被局部于管程内的过程访问。

2）一个进程只有通过调用管程内的过程才能进入广成访问的共享数据。

3）每次仅允许一个进程在管程内执行某个内部过程。

由于管程是一个语言成分，所以管程的互斥访问完全由编译程序在编译时自动添加，无需程序员关注，而且保证正确。

5、经典同步问题

问题描述：一组生产者进程和一组消费者进程共享一个初始为空、大小为n的缓冲区，只有缓冲区没满时，生产者才能把消息放入到缓冲区，否则必须等待；只有缓冲区不为空时，消费者才能从中取出消息，否则必须等待。由于缓冲区是临界资源，它只允许一个生产者放入消息，或一个消费者从中取出消息。

问题分析：

1）关系分析。生产者和消费者对缓冲区互斥访问是互斥关系，同时生产者和消费者又是一个相互协作的关系，只有生产者生产之后，消费者才能消费，他们也是同步关系。

2）整理思路。这里比较简单，只有生产者和消费者两个进程，正好是这两个进程存在着互斥关系和同步关系。那么需要解决的是胡吃喝同步PV操作的位置。

3）信号量的设置。信号量mutex作为互斥信号量，它用于控制互斥访问缓冲池，互斥信号量初始为1；信号量full用于记录当前缓冲池中“满”缓冲区数，初值为0.信号量empty用于记录当前缓冲池中空缓冲区，初值为n。

下面再看一个较为复杂的生产者-消费者问题：

问题描述：桌子上有一只盘子，每次孩子能向其中放入一个水果。爸爸专向盘子中放入苹果，妈妈专向盘子中放入橘子，女儿专等吃盘子中的苹果，儿子专等吃盘子中的橘子。只有盘子为空时，爸爸或妈妈就可向盘子中放一只水果2；仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出。

问题分析：

1）关系分析。这里的关系略微复杂一些，首先由每次只能向其中放入一只水果可知爸爸和妈妈是互斥关系。爸爸和女儿、妈妈和儿子是同步关系，而且这两对进城必须连起来，儿子和女儿之间没有互斥和同步关系，因为他们是选择条件执行，不可能并发。

2）整理思路。这里有4个进程，实际上可以抽象为两个生产者和两个消费者被连接到大小为1的缓冲区上。

3）信号量设置。首先设置信号量plate为互斥信号量，表示是否允许想盘子放水果，初值为1，表示允许放入，且只允许放一只。信号量apple表示盘子里是否有苹果，初值为0，表示盘子中无2苹果；信号量orange表示盘子中是否有橘子，初始量为0，表示盘子中无橘子。

问题描述：有读者和写者两组并发进程，共享一个文件，当两个以上的读进程同事访问共享数据时不会产生副作用，但若某个写进程和其他进程（读进程或写进程）同时访问共享数据时则可能导致数据不一致的错误。因此要求：1）允许多个读者同时对文件执行读操作；2）只允许一个写者往文件中写信息；3）任一写者在完成写操作之前不允许其他读者或写者工作；4）写者执行完写操作前，应让已有的读者或写者全部退出。

问题分析：

1）关系分析。由题目分析读者和写者是互斥的，写者和写者也是互斥的，而读者和读者不存在互斥关系。

2）整理思路。两个进程，即读者和写者。写者比较简凡，它和任何进程互斥，用互斥信号量p操作、v操作即可解决。读者的问题比较复杂，它必须实现与写着互斥的关系，还要实现和其他读者同步的关系。因此，紧急简单的一对pv操作时无法解决的。那么在这里用到了一个计数器，用它来判断当前是否有读者读文件。当有读者的时候，写着是无法写文件的，此时读者会一直占用文件，当没有读者的时候，写者才可以写文件。同事这里不同读者对计数器的访问也是互斥的。

3）信号量的设置。首先设置信号量count为计数器，用来记录当前读者数量，初值为0；设置mutex为互斥信号量，用于保护更新count变量时的互斥；设置互斥信号量rw用于保证读者和写者的互斥访问。

问题描述：一张圆桌上坐着五个哲学家，每两个哲学家之间的桌子上摆着一根筷子，桌子的中间是一碗米饭。哲学家们倾注毕生精力用于思考和进餐，哲学家在思考是，并不影响其他人。只有当哲学家饥饿的时候，才视图拿起左右两根筷子。如果筷子已经在他人手上，则需等待。饥饿的哲学家只有同时拿到了两根筷子才可以开始进餐，当今参悟你比猴，放下筷子继续思考。

问题分析：

1）关系分析。五个哲学家与左右邻居对其中的筷子的访问是互斥关系。

2）整理思路。显然这五个进程，要解决的问题有两个：一个是让他们同时拿起两个筷子；而是对每个哲学家的动作执行规则，避免饥饿或者死锁现象发生。

3）信号量设置。定义互斥信号组chopsticks【5】={1,1,1,1,1}用于对五个筷子的互斥访问。

对哲学家按顺序0~4编号，哲学家i左边的筷子编号为i，哲学家右边的筷子编号为（i+1）%5。

问题描述：假设一个系统有三个吸烟者进程和一个供应者进程。每个抽烟者不停的卷烟并抽调他，但是要卷起并抽掉一支烟，抽烟者必须要有三种材料：烟草、纸和胶水。三个抽烟者中，第一个有烟草，第二个有纸，第三个有胶水。供应者进程无线地提供提供三种材料。

供应者每次将两种材料放到桌子上，拥有剩下那种材料的抽烟者卷一根烟并抽掉它，并给供应者一个信号告诉完成了，供应者就会放另外两种材料在桌子上，这种过程一直重复。

问题分析：

1）关系分析。供应者与三个抽烟者分别是同步关系。由于供应者无法同时满足两个或两个以上的抽烟者，三个抽烟者对抽烟这个动作是互斥关系。

2）整理思路。显然这里有四个进程。供应者作为生产者向三个抽烟者提供材料。

3）信号量设置。信号量offer1、offer2、offer3分别表示烟草和纸组合的资源、烟草和胶水组合的资源、纸和胶水组合的资源。信号量finish 用于互斥进行抽烟动作。

2.4、死锁

1、死锁的概念

在多道程序系统中，由于多个进程的并发执行，改善了系统资源的利用率并提高了系统的处理能力。然而，多个进程的并发执行也带来了新的问题——死锁。所谓死锁是指多个进程因竞争资源而造成的一种僵局，若无外力作用，这些进程都将无法向前推进。

1）系统资源的竞争

通常系统中拥有的不可剥夺资源，其数量不足一满足多个进程运行的需要，似的进程在运行过程中会因争夺资源而陷入僵局。只有对不可剥夺资源的竞争才可能产生死锁，对可剥夺资源的竞争是不会引起死锁的。

2）进程推进顺序非法

进程在运行过程中，请求和释放资源的顺序不当，同样会导致死锁。

信号量使用不当也会造成死锁。进程间相互等待对方发来的消息，结果也会造成某些进程间无法继续向前推进。

3）死锁产生的必要条件

产生死锁必须同时满足以下四个条件，只要其中任一个条件不成立，死锁就不会发生。

互斥条件：进程要求对所分配的资源进行排他性控制，即在一段时间内某资源仅为一个进程所占用。此时若有其他进程请求该资源，则请求进程只能等待。

不剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放。

请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

循环等待条件：存在一种进程资源的循环等待链，连中每一个进程已获得的资源同时被链中下一个进程所请求。

2、死锁的处理策略

为使系统不发生死锁，必须设法破坏产生死锁的四个必要条件之一，或者允许死锁产生，但当死锁发生时能检测出思索，并有能力实现恢复。

死锁处理策略有：

破坏死锁的四个必要条件之一。

用某种方式防止系统进入不安全状态。

允许进程在运行过程中发生死锁，通过系统的检测机构及时地检测出死锁的发生，然后采取某种措施解除死锁。

防止死锁发生只需要破坏死锁产生的四个必要条件之一即可。

允许系统资源都能共享使用。（不太可行）

当一个以保持了某些不可剥夺资源的进程，请求新的资源时得不到满足，它必须释放已经保持的所有资源，待以后需要时再重新申请。这种方法常用于状态易于保存和恢复的资源，如CPU的寄存器及内存资源，一般不能用于打印机之类的资源。

采用预先静态分配方法， 即进程在运行前一次申请完他所需要的全部资源，在他的资源未满足前，不把它投入运行。一旦运行后，这些资源就一直归它所有，也不再提出其他资源请求，这样就可以保证系统不会发生死锁。

这种方式实现简单，但缺点也显而易见，系统资源被严重浪费，其中有些资源可能仅在运行初期或末期才使用，甚至根本不使用。而且还会导致“饥饿”现象，当由于个别资源长期被个别资源占用时，将只是等待该资源的进程迟迟不能开始运行。

（4）破坏循环等待条件

为了破坏循环等待条件，可采用顺序资源分配法。首先给系统中的资源编号，规定每个进程，必须按编号递增的顺序请求资源，同类资源一次申请完。也就是说，只要进程提出申请分配资源，则该进程在以后的资源申请中，只能申请编号比之前大的资源。

4、死锁避免

避免思索同样是属于事先预防的策略，但并不是事先采取某种限制措施破坏死锁的必须条件，而是在资源动态分配过程中，防止系统进入不安全状态，以避免死锁发生。这种方法所施加的限制条件较弱，可以获得较好的系统性能。

避免死锁的方法中，允许进程动态的申请资源，但系统在进行资源分配前，应先计算此次资源分配的安全性。若此次分配不会导致系统进入不安全状态，则将资源你分配给进程，否则，让进程等待。

所谓安全状态，是指系统能按某种进程推进顺序，为每个进程分配其所需的资源，直至满足每个进程对资源的最大需求，是每个进程都可以顺序的完成。此时成P1P2P3。。为安全序列，如果系统无法找到一个安全序列，则称系统处于不安全状态。

并非所有的不安全状态都是死锁状态，但当系统进入不安全状态后，便可能进入死锁状态；反之，只要系统处于安全状态，系统便可以避免进入死锁状态。

银行家算法是最著名的死锁避免算法。

它提出的思想是：把操作系统看作是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。操作系统按照银行家制定的规则为进程分配资源，当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足他的最大需求量，则按当前的申请量分配资源，否则就推迟分配。当进程在执行中继续申请资源时，先测试该进程已占用的资源数与本次申请的资源数之和是否超过了该进程对资源的最大需求量。若超过，则拒绝分配资源，若没有超过则在测试系统现存的资源能否满足该进程尚需的最大资源量，若能满足则按当前的申请量分配资源，否则也要推迟分配。

1）数据结构描述：

可利用资源适量Available：含有m个元素的数组，其中的每一个元素代表一个可用的资源数目，Aviailable【j】=K，则表示系统中现有Rj类资源K个。

最大需求矩阵Max：为n*m矩阵，定义了系统中n个进程中的每个进程中对m类资源的最大需求。Max【ij】=K，则表示进程i需要Rj类资源的最大数目为K。

分配矩阵Allocation：为n*m矩阵，定义了系统中每一类资源当前已分配给每一进程的资源数。Allocation【ij】=K，则表示进程i当前已分得Rj类资源的数目为K。

需求矩阵Need：为n*m矩阵，表示每个进程尚需的各类资源数。Need【ij】=K，则表示进程i还需要Rj类资源数目为K。

上述三个矩阵间存在下述关系：

Need【ij】=Max【ij】-Allocation【ij】

2）银行家算法描述：

设Requesti表示进程Pi的请求适量，如果Requesti【j】=K

，表示进程Pi需要Rj类资源K个。当Pi发出资源请求后，系统按下述步骤进行检查：

1如果Requesti[j]<=Need[I,j],便转向步骤2；否则，认为出错，因为它所需要的资源已超过它所宣布的最大值。

2如果Requesti【j】<=Available【j】，便转向步骤3；否则，表示尚无足够资源，Pi需等待。

3系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：

Available【j】=Available【j】-Requesti【j】；

Allocation【ij】=Allocation【ij】+Requesti【j】；

Need【ij】=Need【ij】-Requesti【j】；

4系统执行安全性算法，检查此次西苑分配后，系统是否出于安全状态，若安全，才正式把资源分配给进程Pi，已完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程Pi等待。

3）安全性算法

1设置两个矢量。工作矢量work：它表示系统可供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，work=Available；

Finish：它表示系统是否有足够的资源分配给进程，使之运行完成。开始时，Finish【i】=false；当有足够资源分配给进程Pi时，再令Finish【i】=true。

2从进程集合中找到一个能满足下述条件的进程：Finish【i】=false；Need【ij】<=work【j】；若找到，执行下一步骤，否则，执行步骤4。

3当进程Pi获得资源后，可顺利执行，直至完成，并释放粗分配给它的资源，故应执行：

Work【j】=work【j】+Allocation【ij】；

Finish【i】=true；

Go to step2；

4如果所有进程的Finish【i】=true都满足，则表示系统处于安全状态；否则，系统处于不安全状态。

5、死锁的检测和解除

前面介绍的死锁预防和死锁避免都是在为进城分配资源时施加限制条件或进行检测，若系统为进程分配资源时不采取任何措施，则应该提供 死锁检查和解除的手段。

系统死锁，可利用资源分配图来描述。用圆圈代表一个进程，用方框代表一类资源。由于一种类型的资源可能有很多歌，用框中的一个点代表一类资源中的一个资源。从晋城到资源的有向边叫请求边，表示该进程请申请一个单位的该类资源；从资源到进程的边叫做分配边，表示该类资源已经有一个资源被分配到了该进程。

可以通过将资源分配图简化的方法来检测系统状态S是否为死锁状态。简化方法如下：

1）在资源分配图中，找出既不阻塞又不是孤点的进程Pi（即找出一条有向边与它相连，且该有向边对应资源的申请数量小于系统中已有空闲资源数量。若所有的连接该进程的边都满足上述条件，则这个进程能继续运行直至完成，然后释放它所占有的所有资源）。消去它所有的请求边和分配边，使之成为孤立的节点。

2）进程Pi所释放的资源，可以唤醒某些因等待这些资源而阻塞的进程，原来的阻塞进程可能变为非阻塞进程。

S为死锁的条件是当且仅当S状态的资源分配图是不可简化的，该条件为死锁定理。

一旦检测出死锁，则应立即采取相应的措施，已解除死锁。死锁解除的主要方法有：

1）资源剥夺法。挂起某些思索进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源时，而处于资源匮乏的状态。

2）进程撤销法。强制撤销一个或一部分进程并剥夺这些进程的资源。撤销的原则可以按进程的优先级和撤销进程代价的高低进行。

3）进程回退法。让一个或多个进程回退到足以回避死锁的地步，进程回退时资源释放资源而不是被剥夺。要求系统保持进程的历史信息，设置还原点。

2.5、本章疑难点

1、进程与撑血的区别与联系

2、死锁与饥饿

具有等待队列的信号量的实现可能导致这样的情况：两个或多个进程无限的等待一个事件，而该事件只能由这些等待进程之一来产生。这里的事件是V操作的执行，当出现这样的状态时，这些进程成为死锁。

说一组集成处于死锁状态是指：组内的每个进程都等待一个事件，而该事件只可能由组内的另一个进程产生。这里关心的主要事件是资源的获取和释放。

与死锁相关的另一个问题是无限期阻塞或“饥饿”，即进程在信号量内无穷等待的情况。

产生饥饿的主要原因是：在一个动态系统中，对于每类系统资源，操作系统需要确定一个分配策略，当多个进程同时申请某类资源是，由分配策略确定资源分配给进程的次序。有的时候资源分配策略可能是不公平的，既不能保证等待时间上界的存在。在这种情况下，及时系统没有发生死锁，某些进程也可能会长时间等待。当等待时间给进程推荐和响应带来明显影响时，城发生了进程“饥饿”，当“饥饿”到一定程度的进程所赋予的任务即使完成也不再具有实际意义时就成该进程被“饿死”。

“饥饿”并不表示系统一定死锁。但至少有一个进程的执行被无限期的推迟，“饥饿”与死锁的主要区别有：

1）进入饥饿状态的进程可以只有一个，但由于循环等待条件进入死锁状态的进程却必须大于或等于两个。

2）处于饥饿状态的进程可以处于就绪状态，而处于死锁状态的进程则必定是处于阻塞状态。

3、银行家算法的工作原理

银行家算法的主要思想是避免系统进入不安全状态。在每次进行资源分配时，它首先检查系统是否有足够的资源满足要求，如果有，则先进行分配，并对分配后的新状态进行安全性检查。如果新状态安全，则正式分配上述资源，否则就拒绝分配上述资源。这样，它保证系统始终处于安全状态，从而避免死锁现象的发生。

4、进程同步、互斥的区别和联系

并阿发进程的执行会产生想制约的关系：一种是进程之间竞争使用临界资源，只能让它们逐个使用，这种现象称为互斥，是一种竞争关系；另一种是进程之间协同完成任务，在关键点上等待另一个进程发来的消息，以便协同一致，是一种协作关系。

5、作业和进程的关系

进程是系统资源的使用者，系统的资源大部分都是以进程为单位分配的。而用户使用计算机是为了实现一串相关的任务，通常把用户要求计算机完成的这一串任务称为作业。

批处理系统中的可以通过磁记录设备或系统提交作业，由系统的SPOOLLing输入进程将作业放入磁盘的输入井中，作为后备作业。作业调度程序（一般也作为独立的进程运行）每选择一刀后备作业运行时，首先为该作业创建一个进程（称为该作业的根进程）。该进程将执行作业控制语言解释程序

第三章 内存管理

3.1、内存管理基础

1、内存管理的概念

内存管理是操作系统设计中最重要和最复杂的内容之一。计算机硬件一直在发展，内容容量也在不断增长，但是仍然不可能将所有用户进程和系统所需要的全部程序和数据全部放入主存中，所以操作系统必须将内存空间进行合理的化肥和有效的动态分配。操作系统对内存的划分和动态分配，就是内存管理的概念。

有效的内存管理在多道程序设计中非常重要，不仅方便用户使用存储器、提高内存利用率，还可以通过虚拟技术从逻辑上扩充存储器。

内存管理的功能有：

l 内存空间的分配与回收，包括内存的分配和共享。

l 地址转换，把逻辑地址转换成相应的物理地址。

l 内存空间的扩充，利用虚拟技术或自动覆盖技术，从逻辑上扩充内存。

l 存储保护，保证各道作业在各自存储空间内运行，互不干扰。

在进行具体的内存管理之前，需要了解进程运行的基本原理和要求。

创建进程首先要将程序和数据装入内存。将用户原程序变成可在内存中执行的程序，通常需要以下几个步骤。

l 编译，由编译程序将用户源代码编译成若干个目标模块。

l 链接，由链接程序将编译后形成的一组目标模块，以及所需库函数链接，形成完整的装入模块。

l 装入，由装入程序将装入模块装入内存。

程序的链接有以下三种方式：

l 静态链接：在程序运行之前，先将各目标模块及它们所需的库函数链接成一个完整的可执行程序，以后不再拆开。

l 装入时动态链接：将用户源程序编译后所得到的一组目标模块，再装入内存时，采用边装入变链接的方式。

l 运行时动态链接：对某些目标模块的连接，是在程序执行中需要该目标模块时，才对她进行链接。其优点是便于修改和更新，便于实现对目标模块的共享。

内存的装入模块再装入内存时，同样有以下三种方式：

1）绝对装入。在编译时，如果知道程序将驻留在内存的某个位置，编译程序将产生绝对地址的目标代码。绝对装入程序按照装入模块的地址，将程序和数据装入内存。装入模块被装入内存后，由于程序中的逻辑地址与实际地址完全相同，故不需对程序和数据的地址进行修改。

绝对装入方式只适用于单道程序环境。另外，程序中所使用的绝对地址，可在编译或汇编时给出，也可由程序员直接赋予。

2）可重定位装入。在多道程序环境下，多个目标模块的起始地址通常都是从0开始，程序中的其他地址都是相对于起始地址的，此时应采用可重定位装入方式。根据内存的当前情况，将装入模块装入到内存的适当位置。装入时对目标程序中指令和数据的修改过程称为重定位，地址变换通常是装入时一次完成，所以成为静态重定位。

其特点是在一个作业装入内存时，必须分配器要求的全部内存空间，如果没有足够的内存，就不能装入，此外一旦作业进入内存后，在整个运行期间，不能在内存中移动，也不能再申请内存空间。

3）动态运行时装入，也成为动态重定位，程序在内存中如果发生移动，就需要采用动态的装入方式。

动态运行时的装入程序在把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是把这种地址转换推迟到程序眞正要执行时才进行。因此，装入内存后的所有地址均为相对地址，这种方式需要一个重定位寄存器的支持。

其特点是可以将程序分配到不连续的存储区中；在程序运行之前可以只装入它的部分代码即可运行，然后在程序运行期间，根据需要动态申请分配内存；便于程序段的共享，可以向用户提供一个比存储空间大得多的地址空间。

编译后，一个目标程序所限定的地址范围称为改程序的逻辑地址空间。编译程序在对一个源程序进行编译时，总是从0号单元开始为期分配地址，地址空间中的所有地址都是相对起始地址0的，因而逻辑地址也称为相对地址。用户程序和程序员只需要知道逻辑地址，而内存管理的具体机制则是透明的，这些只有系统编程人员才会涉及。不同进程可以有相同的逻辑地址，因为这些相同的逻辑地址可以映射到主存的不同位置。

物理地址空间实质内存中物理单位的集合，它是地址转换的最终地址，进程在运行时执行指令和访问数据最后都要通过物理地址来存取主存。当装入程序将可执行代码装入内存时，必须通过地址转换将逻辑地址转换成物理地址，这个过程称为地址重定位。

内存分配前，需要保护操作系统不受用户进程的影响，同时保护用户进程不受其他用户进程的影响。通过采用重定位寄存器和界地址寄存器来实现这种保护。重定位寄存器含最小的物理地址值，界地址寄存器含逻辑地址值。每个逻辑地址值必须小于界地址寄存器。内存管理机构动态地将逻辑地址加上重定位寄存器的值后映射成物理地址，再送交内存单元。

当CPU调度程序选择进程执行时，派遣程序会初始化重定位寄存器和界地址寄存器。每个地址都需要与寄存器进行核对，可以保证操作系统和其他用户程序及数据不被该进程运行所影响。

2、覆盖与交换

覆盖与交换技术是在多道程序环境下用来扩充内存的两种方法。覆盖技术主要用在早期的操作系统中，而交换技术则在现代操作系统中仍具有较强的生命力。

早期的计算机系统中，主存容量很小，虽然住村中仅存放一道用户程序，但是存储空间放不下用户进程的现象也经常发生，这一矛盾可以用覆盖技术来解决。其基本思想是：由于程序运行时并非任何时候都要访问程序和数据的各个部分，因此可以把用户空间分成一个固定区和若干个覆盖区。将经常活跃的部分放在固定区，其余部分按调用关系分段。首先将那些将要访问的段放入覆盖区，其他段放在外存中，在需要调用时，系统再将其掉入覆盖区，替换其中原有的段。

交换的基本思想是：把处于等待状态（或在CPU调度原则下被剥夺运行权利）的进程从内存移到辅存，把内存空间腾出来，这一过程又叫换出；把准备好竞争CPU运行的进程从辅存移到内存，这一过程又称为换入。

例如，有一个CPU采用时间片轮转调度算法的多道程序环境。时间片到，内存管理器将刚刚执行过的进程换出，将另一进程换入到刚刚释放的内存空间中。同时，CPU调度器可以将时间片分配给其他已在内存中的进程。每个进程用完时间片都与另一进程交换。理想情况下。内存管理器的交换过程速度足够快，总有进程在内存中可以执行。

有关交换需要注意以下几个问题：

l 交换需要备份存储，通常是快速磁盘。它必须足够大，并且提供对这些内存影响的直接访问。

l 为了有效使用CPU，需要每个进程的执行时间比交换时间长，而影响交换时间的主要是转移时间。转移时间与所见换的内存空间成正比。

l 如果换出进程，必须确保该进程是完全处于空闲状态。

l 交换空间通常作为磁盘的一整块，且独立与文件系统，因此使用就可能很快。

l 交换通常在有许多进程运行且内存空间吃紧的时候开始启动，而系统负荷降低就暂停。

l 普通的交换使用不多，但交换策略的某些变种在许多系统中仍发挥作用。

交换技术主要是在不同进程之间进行，而覆盖则用于同一个程序中。由于覆盖技术要求给程序段之间的覆盖结构，使得其对用户和程序员不透明，所以对于主存无法存放用户程序的矛盾，现在操作系统是通过虚拟内存技术来解决的，覆盖技术则已成为历史；而交换技术在现代操作系统中仍具有较强的生命力。

3、连续分配管理方式

连续分配方式，是指为一个用户程序分配一个连续的内存空间。它主要包括单一连续分配、固定分区分配和动态分区分配。

内存在此方式下分为系统区和用户区，系统区仅提供给操作系统使用，通常在低地址部分；用户区是为用户提供的除系统外的内存空间。这种方式无需进行内存保护。

这种方式的优点是简单、无外部碎片，可以采用覆盖技术，不需要额外的技术支持。缺点是只能用于单用户、单任务的操作系统中，有内部碎片，存储器的利用率极低。

固定分区分配是最简单的一种多道程序存储管理方式，它将内存用户空间划分为若干个固定大小的区域，每个分区只装入一道作业。当有空闲分区时，便可以再从外存的后备队列中选择适当大小的作业装入该分区。如此循环。

固定分区分配在划分分区时，有两种不同的方法：

l 分区大小相等：用于利用一台计算机去控制多个相同对象的场合。

l 分区大小不等：划分为含有多个较小的分区、适量的中等分区及少量的大分区。

为了便于内存分配，通常将分区按大小排队，并为之建立一张分区使用表，其中个表项包括每个分区的起始地址、大小及状态。当有用户程序要装入时，便检索该表，已找到合适的分区给与分配并将其状态置为“已分配“。未找到合适分区则拒绝为该用户程序分配内存。

这种分区方式存在两个问题：一个程序可能太大而放不进任何一个分区中，这是用户不得不使用覆盖技术来使用内存空间；二是主存利用率低，当程序小于固定分区大小时，也占用了一个完整的内存分区空间，这样分区内部有空间浪费。这种现象成为内部碎片。

固定分区可用于多道程序设计最简单的存储分配，但不能实现多进程共享一个主存区，所以存储空间利用率低。固定分区分配很少用于现在通用的计算机，但在某些用于控制多个相同对象的控制系统中仍发挥着一定的作用。

动态分区分配又称为可变分区分配，是一种动态划分内存的分区方法。这种分区方法预先将内存划分，而是在进程装入内存时，根据进程的大小动态的建立分区，并使分区的大小正好适合进程的需要。因此系统中分区的大小和数目是可变的。

动态分区在开始分配时是很好的，但是之后会导致内存中出现许多小的内存块。随着时间的推移，内存中会产生越来越多的碎片，内存的利用率随之下降。这种现象称之为外部碎片现象，指在所有分区外的存储空间会变成越来越多的碎片，这与固定分区中的内部碎片正好相对。克服外部碎片可以通过紧凑技术来解决，就是操作系统不时地对进程进行移动和整理。但是这需要动态定位的支持，且相对费时。紧凑的过程实际上类似于windows系统中的磁盘整理程序，只不过后者是对外存空间的紧凑。

在津城装入或换入主存时。如果内存中有多个足够大的空闲块，操作系统必须确定分配那个内存块给进程使用，这就是动态分区的分配策略。，考虑以下几种算法：

1）首次适应算法：空闲分区以地址递增的次序链接。分配内存时顺序查找，找到大小能满足要求的第一个空闲分区。

2）最佳适应算法：空闲分区按容量递增形成分区链，找到第一个能满足要求的空闲分区。

3）最坏适应算法：有称最大适应算法，空闲分区以容量递减次序链接。找到第一个能满足要求的空闲分区，也就是挑选最大的分区。

4）临近适应算法：又称循环首次适应算法，由首次适应算法演变而成。不同之处是分配内存时从此查找结束的位置开始继续查找。

在这几种方法中，首次适应算法不仅是最简单的，而且通常是最好和最快的。在UNIX系统的最初版本中，就是使用首次适应算法为进程分配内存空间，其中使用数组的数据结构（而非链表）来实现。不过，首次适应算法会使得内存的低地址部分出现很多小的空闲分区，而每次分配查找时，都要经过这些分区。

临近适应算法试图解决这一问题，但实际上，它常常会导致在内存的末尾分配空间，分裂成小碎片。它通常比首次适应算法的结果要差。

最佳适应算法虽然称为最佳，但是性能通常很差，因为每次最佳的分配会留下最小的内存块，它会产生最多的碎片。

最坏适应算法与最佳适应算法相反，选择最大的可用块，这看起来最不容易产生碎片，但是却把最大的连续内存划分开，会很快导致没有可用的大的内存块，因此性能非常差。

以上内存分区管理方法有一共同特点，即用户进程在主存中都是连续存放的。

4、非连续分配管理方式

非连续分配方式允许一个程序分散的装入不相邻的内存分区中，根据分区的大小是否固定分为分页存储管理方式和分段存储管理方式。

分页存储管理方式中，又根据运行作业时是否要把作业的所有页面都装入内存才能运行分为基本分页存储管理和请求分页存储管理方式。

固定分区会产生内部碎片，动态分区会产生外部碎片，两种技术对内存的利用率都比较低。我们希望内存的使用能尽量避免碎片的产生，这就引出了分页思想：把主存空间划分为大小相等且固定的块，块相对较小，作为主存的基本单位。每个进程也以块为单位进行划分，进程在执行时，以块为单位逐个申请主存中的块空间。

1）分页存储的几个基本概念

1页面和页面大小。进程中的块称为页，内存中的块称为页框。外存也以同样单位划分，直接称为块。进程在执行时需要申请主存空间，就是要为每个页面分配主存中的可用页框，这就产生了页和页框的一一对应。

为了方便地址转换，页面大小应是2的整数幂。同时页面大小应当适中。如果页面太小，会是进程的页面数过多，这样页表就过长，占用大量内存，而且也会增加硬件地址转换的开销，降低页面换入换出的效率。页面过大又会是页面碎片过大，降低内存利用率。所以页面的大小应该适中，考虑到空间效率和时间效率。

2地质结构。分页存储管理的地质结构包含两部分：前一部分为页号，后一部分为页内偏移量W。地址长度为32位，其中0~11为页内地址，即每页大小为4kB；12~31位为页号，地址空间最多允许有2 20页。

3页表。为了便于在内存中找到进程的每个页面所对应的物理块，系统为每个进程建立一张页表，记录页面在内存中对应的物理块号，页表一般存放在内存中。

在配置了页表后，进程执行时，通过查找该表，即可找到每页在内存中中的物理块号。可见，页表的作用是实现从页号到物理块号的地址映射。

2）基本地址变换机构

地址变换机构的任务是将逻辑地址中的页号，转换为内存中物理块号，地址变换是借助于页表实现的。

在系统中通常设置一个页表寄存器PTR，存放页表在内存的初值和页表长度。

逻辑地址到物理地址的变换过程如下：

1地址变换机构自动将有效地址分为页号和页内偏移量两部分，再用页号去检索页表。在执行检索之前，先将页号与页表长度比较，如果页号大于或等于页表长度，则表示地址越界并中断。

2若未越界，则将页表始址与页号和页表项长度的乘积相加，便得到该表项在页表中的位置，于是可从中得到该页的物理块号。

3与此同时，在将有效地址中的页内偏移量送入物理地址寄存器的块内地址字段中。

以上整个地址变换过程均是由硬件自动完成的。

下面讨论分页管理方式存在的两个主要问题：1每次访存操作都需要进行逻辑地址到物理地址的转换，地址转换过程必须足够快，否则访存速度会降低；2每个进程引入了页表，用于存储映射机制，页表不能太大，否则内存利用率会降低。

3）具有快表的地址变换机构

由上面介绍的地址变换过程可知，若页表全部放在内存中，则要存取一个数据或一条指令至少要访问两次内存：一次是访问页表，确定要存取的数据或指令的物理地址，第二次才根据该地址存取数据或指令。显然，这种方法比通常执行指令的速度慢了一半。

为此，在地址变换机构中增设了一个具有并行查找能力的高速缓冲存储器——快表，又称联想寄存器TLB，用以存放当前访问的若干页表项。与此对应，主存中的页表也常称为慢表。

在具有快表的分页机制中，地址的变换过程：

1CPU给出有效地址后，由硬件进行地址转换，并将页号送入高速缓存寄存器，并将此页号与快表中的所有页号同时进行比较。

2如果有找到匹配的页号，说明索要访问的页表项在快表中，则可以直接从中读出该页对应的页框号，送到屋里地址寄存器。这样存取数据可以直接一次访存实现。

3如果没有找到，则需要访问主存中的页表，在读出页表项后，应同时将其存入快表中，以供后面可能的再次访问。但是如果快表已满，就必须按照一定的算法对其中旧的页表项进行替换。注意，有些处理器设计为快表和主存同时查找，如果在快表中匹配成功则终止主存中的查找。

一般快表的命中率可以达到90%，这样，分页带来的速度损失就降到10%。快表的有效性是基于著名的局部性原理。这在后面的虚拟内存中将会具体讨论。

4）两级页表

第二个问题：由于引入了分页管理，进程在执行时不需要将所有页调入内存页框中，而只要将保存有映射关系的页表调入内存即可。但是我们仍然需要考虑页表的大小。如果页表太大，肯定是降低了内存利用率的；从另一方面来说，程序所有的页表项也并不需要同时保存在内存中，因为在大多数情况下，映射所需要的页表都再也表的同一个页面中。

我们将页表映射的思想进一步延伸，就可以得到二级分页：将页表的10页空间也进行地址映射，建立上一级页表，所以上一级页表只需要一页就足够。在进程执行时，只需要将这一页上一级页表调入内存即可，进程的页表和进程本身的页面，可以在后面的执行中再调入内存。

分页管理方式是从计算机的角度考虑设计的，以提高内存的利用率，提升计算机的性能，且分页通过硬件机制实现，对用户完全透明；而分段管理方式的提出则考虑了用户和程序员，以满足方便编程、信息保护和共享、动态增长及动态链接等多方面的需要。

1）分段。

短时系统按照用户进程中的自然段划分逻辑空间。例如，用户进程由主程序、两个字程序、栈和一段数据组成，于是可以把这个用户进程划分为5个段，每段从0开始编址，并采用一段连续的地址空间（段内要求连续，段间不要求连续），其逻辑地址由两部分组成：段号与段内偏移量，分别记为S、W。

段号为16位，段内偏移量为16位，则一个作业最多可有2 16=65536个段，最大段长64KB。

在页式系统中，逻辑地址的页号和页内偏移量对用户是透明的；但在段式系统中，段号和段内偏移量必须由用户显示提供，在高级程序设计语言中，这个工作由编译程序完成。

2）段表。

每个进程都有一张逻辑空间与主存空间映射的段表，其中每一段表项对应进程的一个段，段表项纪录路该段在内存中的起始地址和段的长度。

在配置了段表后，执行中的进程可通过查找段表，找到每个段所对应的内存区。可见，段表用于实现从逻辑端段到物理内存区的映射。

3）地址变换机构

为了实现进程从逻辑地址到物理地址的变换功能，在系统中设置了段表寄存器，用于存放段表始址和段表长度TL。在进行地质变换时，系统将逻辑地址中的段号，与段表长度TL比较。若段号大雨段表长度，表示短号太大，访问越界，于是产生越界中断信号。若未越界，则根据段表的始址和该段的段号，计算出该段对应段表项的位置，从中读出该段在内存中的起始地址。然后，在检查段内地址W是否超过该段的段长SL。若超过，同样发出越界中断信号。若未越界，则将该段的基址d与段内地址相加，即可得到要访问的内存物理地址。

页式存储管理能有效的提高内存利用率，而分段存储管理能反应程序的逻辑结构并有利于段的共享。如果将这两种存储管理方法结合起来，就形成了段页式存储管理方式。

在段页式系统中，作业的地址空间首先被分成若干个逻辑段，每段都有自己的段号，然后再将每一段分成若干个大小固定的页。对内存空间的管理仍然和分页存储管理一样，将其分成若干个和页面大小相同的存储块，对内存的分配以存储块为单位。

在段页式系统中，作业的逻辑地址分为三部分：段号、页号和页内偏移量。

为了实现地址变换，系统为每个进程建立一张段表，而每个分段有一张页表。段表表项中至少包括段号、页表长度和页表起始地址，页表表项中至少包括页号和块号。此外，系统中还应有一个段表寄存器，指出作业的段表起始地址和段表长度。

在进行地址变换时，首先通过段表查到页表起始地址，然后通过页表找到帧号，最后形成物理地址。进行一次访问实际需要三次访问主存，这里同样可以使用快表提供加快速度，其关键字由段号、页号组成，值是对应的页帧号和保护码。

3.2、虚拟内存管理

1、虚拟内存的基本概念

上一节所讨论的各种内存管理策略都是为了同时将多个进程保存在内存中以便允许多道程序设计。他们都具有以下两个共同特征：

1）一次性：作业必须一次性全部装入内存后，方可运行。这会导致两种情况发生：1当作业很大，不能全部被装入内存时，将使该作业无法运行；2当大量作业要求运行时，由于内存不足以容纳所有作业，只能使少数作业先运行，导致系统难以运行多道程序。

2）驻留性：作业被装入内存后，就一直驻留在内存中，其任何部分都不会被换出，直至作业运行结束。运行中的进程，会因等待IO而被阻塞，可能处于长期等待状态。

由上分析可知，许多在程序运行中不用或暂时不用的程序（数据）占据了大量的内存空间，而一些需要运行的作业又无法装入运行，显然浪费了宝贵的内存空间。

要真正理解虚拟内存技术的思想，首先必须了解计算机中著名的局部性原理。著名的Bill Joy说过：“在研究所的时候， 我经常开玩笑的说高速缓存是计算机科学中唯一重要的思想。事实上，高速缓存技术确实极大地影响了计算机系统的设计。”快表、页高速缓存以及虚拟内存技术从广义上讲，都是属于高速缓存技术。这个技术所依赖的原理就是局部性原理。局部性原理既适用于程序结构，也适用于数据结构。

局部性原理表现在以下两个方面：

1）时间局部性。如果程序中的某条指令一旦执行，则不久以后该指令可能再次执行；如果某数据被访问过，则不久以后该数据可能再次被访问。产生时间局部性的典型原因，是由于在程序中存在着大量的循环操作。

2）空间局部性。一旦程序访问量某个存储单元，在不久之后，其附近的存储单元也将被访问，即程序在一段时间内所访问的地址，可能集中在一定的范围之内，其典型情况便是程序的顺序执行。

时间局部性是通过将进来使用的指令和数据保存到高速缓存存储器中，并使用高速缓存的层次结构实现。空间局部性通常是使用较大的高速缓存，并将预取机制集成到高速缓存控制逻辑中实现。虚拟内存技术实际上就是建立了“内存-外存”的两级存储器的结构，利用局部性原理实现高速缓存。

基于局部性原理，在程序装入时，可以将程序的一部分装入内存，而将其与部分留在外存，就可以启动程序执行。在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存，然后继续执行程序。另一方面，操作系统将内存中暂时不使用的内容换到外存上，从而腾出空间存放将要调入内存的信息。这样，计算机好像为用户提供了一个比实际内存大的多的存储器，成为虚拟存储器。

之所以将其称为虚拟存储器，是因为这种存储器实际上并不存在，只是由于系统提供了部分装入、请求调入和置换功能后，给用户的感觉是好像存在一个比实际物理内存大得多的存储器。虚拟存储器有以下三个主要特征：

1）多次性，是指无需在作业运行时一次性地全部装入内存，而是允许被分成多次调入内存运行。

2）对换性，是指无需在作业运行时一直常驻内存，而是允许在作业的运行过程中，进行换进和换出。

3）虚拟性 ，是指从逻辑上扩充内存的容量，是用户所看到的内存容量，远大于实际的内存容量。

虚拟内存中，允许讲一个作业分多次调入内存。采用连续分配方式时，会是相当一部分内存空间都处于暂时或永久的空闲状态，造成内存资源的严重浪费，而且也无法从逻辑上扩大内存容量。因此，虚拟内存的实现需要建立在离散分配的内存管理方式的基础上。

虚拟内存的实现有以下三种方式：

l 请求分页存储管理

l 请求分段存储管理

l 请求段页式存储管理

不管哪种方式，都需要有一定的硬件支持。一般需要的支持有以下几个方面：

l 一定容量的内存和外存。

l 页表机制或段表机制，作为主要的数据结构。

l 中断机构，当用户程序要访问的部分尚未调入内存，则产生中断。

l 地址变换机构，逻辑地址到物理地址的变换。

2、请求分页管理方式

请求分页系统建立在基本分页系统基础上，为了支持虚拟存储器功能而增加了请求掉页功能和页面置换功能。请求分也是目前最常用的一种实现虚拟存储器的方法。

在请求分页系统中，只要求将当前需要的一部分页面装入内存，便可以启动作业运行。在作业执行过程中，当所要访问的页面不在内存时，再通过掉页功能将其调入，同时还可以通过置换功能将暂时不用的页面换出到外存上，以便腾出内存空间。

为了实现请求分页，系统必须提供一定的硬件支持。除了需要一定容量的内存及外存的计算机系统，还需要有页表机制、缺页中断机构和地址变换机构。

页表机制不同于基本分页系统，请求分页系统在一作业运行之前不要求全部一次性调入内存，因此在作业的运行过程中，必然会出现要访问的页不在内存的情况，如何发现和处理这种情况是请求分页系统必须解决的两个基本问题。为此，在请求页表项中增加了四个字段：状态位P、访问字段A、修改位M、外存地址。

增加的四个字段说明如下：

状态位P：用于指示该页是否已调入内存，共程序访问时参考。

访问字段A：用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供置换算法换出页面时参考。

修改位M：表示该页在调入内存后是否被修改过。

外存地址：用于指出该也在外存上的地址，通常是物理块号，供调入该页时参考。

在请求分页系统中，每当所要访问的页面不在内存时，便产生一个缺页中断，请求操作系统将所缺的页调入内存。缺页中断作为中断同样要经历诸如：保护CPU环境、分析中断原因、转入缺页中断处理程序进行处理、恢复CPU环境等几个步骤。但与一般的中断相比，它有以下两个明显的区别：

在指令执行期间产生和处理中断信号，而非一条指令执行完后。

一条指令在执行期间，可能产生多次缺页中断。

请求分页系统中的地址变换机构，是在分页系统地址变换机构的基础上，为实现虚拟内存，又增加了某些功能而形成的。

在进行地质变换时，先检索快表：

若找到要访问的页，边修改页表中的访问位，然后利用页表项中给出的物理块号和页内地址形成物理地址。

若为找到该页的页表项，应到内存中去查找页表，在对比页表项中的状态位P，看该页是否已调入内存，未调入则产生缺页中断，请求从外存把该页调入内存。

3、页面置换算法

进程运行时，若其访问的页面不在内存而需将其调入，但内存已无空闲空间时，就需要从内存中调出一页程序或数据，送入磁盘的兑换区。而选择调出页面的算法就成为页面置换算法。好的页面置换算法应有较低的页面更换频率，也就是说，应将以后不会在访问或者以后较长时间内不会访问的页面先调出。

常见的置换算法有以下四种：

最佳置换算法所选择的被淘汰页面将是以后永不使用的，或者是在最长时间内不再被访问的页面，这样可以保证获得最低的缺页率。但由于人们目前无法预知进程在内存下的若干页面中那个是未来最长时间内不再被访问的，因而该算法无法实现。

优先淘汰最早进入内存的页面，亦即在内存中驻留时间最久的页面。该算法实现简单，只需把调入内存的页面根据先后次序连接成队列，设置一个指针总指向最先的页面。但该算法与进程实际运行时的规律不适应，因为在进程中，有的页面经常被访问。

选择最近最长时间未访问过的页面予以淘汰，他认为过去一段时间内未访问过的页面，在最近的将来可能也不会被访问。该算法为每个页面设置一个访问字段，来记录页面自上次被访问以来所经历的时间，淘汰页面时选择现有页面中值最大的予以淘汰。

LRU性能较好，但需要寄存器和栈的硬件支持。LRU是堆栈类的算法。理论上可以证明，堆栈类算法不可能出现Belady异常。FIFO算法基于队列实现，不是堆栈类算法。

LRU算法的性能接近于OPT，但实现起来比较困难，且开销大；FIFP算法实现简单，但性能差。所以操作系统的设计者尝试了很多算法，试图用比较小的开销接近LRU的性能，这类算法都是CLOCK算法的变体。

简单的CLOCK算法是给每一帧关联一个附加位，称为使用位。当某一页首次装入主存时，该帧的使用位设置为1；当该页随后再被访问到时，他的使用位也被置为1.对于液体换算法，用于替换的候选帧集合看做一个循环缓冲区，并且有一个指针与之相关联。当某一页被替换时，该指针被设置成指向缓冲区中的下一帧。当需要替换一页时，操作系统扫描缓冲区，以查找使用位被指为0的帧，每当遇到一个使用位为1的帧时，操作系统就将该位重新置为0；如果在这个过程开始时，缓冲区中所有帧的使用位均为0，则选择遇到的第一个帧替换；如果所有帧的使用位均为1，则指针在缓冲区中完整的循环一周，把所有使用位都置为0，并且停留在最初的位置上，替换该帧中的页。由于该算法循环的检查各页面的情况，故称为CLOCK算法，又称为最近未用NRU（ Not recently used）算法。

CLOCK算法的性能比较接近LRU，而通过增加使用的位数目，可是使得CLOCK算法更加高效。在使用位的基础上再增加一个修改位，则得到改进型的CLOCK置换算法。这样，每一帧都出于以下四种情况之一。

1）最近未被访问，也未被修改（u=0，m=0）。

2）最近被访问，但未被修改（u=1，m=0）。

3）最近未被访问，但被修改（u=0，m=1）。

4）最近被访问，被修改（u=1，m=1）。

算法执行如下操作步骤：

1）从指针的当前位置开始，扫描帧缓冲区。在这次扫描过程中，对使用位不作任何修改，选择遇到的第一个帧（u=0，m=0）用于替换。

2）如果第1步失败，则重新扫描，查找（u=0，m=1）的帧。选额遇到的第一个这样的帧用于替换。在这个扫面过程中，对每个跳过的帧，把它的使用位设置成0.

3）如果第2步失败，指针将回到它的最初位置，并且集合中所有帧的使用位均为0.重复第一步，并且如果有必要重复第2步。这样将可以找到供替换的帧。

改进型的CLOCK算法优于简单的CLOCK算法之处在于替换时首选没有变化的页。由于修改过的页在被替换之前必须写回，因而这样做会节省时间。

4、页面分配策略

对于分页式的虚拟内存，在准备执行时，不需要也不可能把一个进程的所有页都读取到主存，因此，操作系统必须决定读取多少页。也就是说，给特定的进程分配多大的主存空间。这需要考虑以下几点：

1）分配给一个进程的存储量越小，在任何时候驻留在主存的进程数越多，从而可以提高处理器的时间利用率。

2）如果一个进程在主存中的页数过少，尽管有局部性原理，页错误率仍然会相对较高。

3）如果页数过多，由于局部性原理，给特定的进程分配更多的主存空间对该进程的错误率没有明显的影响。

基于这些因素，现代操作系统通常采用三种策略：

1）固定分配局部置换。它为每个进程分配一定数量的物理块，在整个运行期间都不改变。若进程在运行中发现缺页，则只能从该进程在内存的页面中选出一个换出，然后再调入需要的页面。实现这种策略难以确定为每个进程应分配的物理块数量：太少会频繁出现缺页中断，太多又会使CPU和其他资源利用率下降。

2）可变分配全局置换。这是最易于实现的物理块分配和置换策略，为系统中的每个进程分配一定数量的物理块，操作系统自身也保持一个空闲物理块队列。当某进程发现缺页时，系统从空闲物理块队列中取出物理块分配给该进程，并将于调入的页装入其中。

3）可变分配局部置换。它为每个进程分配一定数目的物理块，当某进程发现缺页时，只允许从该进程在内存的页面中选出一页换出，这样就不会影响其他进程的运行。如果进程在运行中频繁的换页，系统需再为该进程分配若干附加物理块，直至该进程缺页率趋于适当程度为止；反之，若一个进程在运行过程中缺页率特别低，则此时可适当减少该进程的物理块。

为确定系统将进程运行时所缺的页面调入内存的时机，可采取预调页策略或请求调页策略。

1）预调页策略。根据局部性原理，一次调入若干个相邻的页可能比一次调入一页更高效。但如果调入的一批页面中大厦多数都未被访问，则又是低效的。所以就需要采用以预测为基础的预调页策略，将预计在不久之后便会被访问的页面预先调入内存。但目前预调页的成功率仅约50%。股这种策略主要用于进程的首次调入时，有程序员指出应该先调入哪些页。

2）请求调页策略。进程在运行中需要访问的页面不在内存而提出的请求，由系统将所需页面调入内存。这种策略调入的页一定会被访问，且这种策略比较易于实现，故在目前的虚拟存储器中大多采用此策略。它的缺点在于每次调入一页，会花费过多的IO开销。

3）从何处调入页面。

请求分页系统中的外存分为两部分：用于存放文件的文件区和用于存放对换页面的对换区。对换区通常是采用连续分配方式，而文件区采用离散分配方式，故对换区的磁盘IO速度比文件区高。这里从何处调入页面有三种情况：

1）系统拥有足够的对换区空间：可以全部从对换区调入所需页面，以提高调页速度。为此，在进程运行前，需将与该进程有关的文件从文件区复制到对换区。

2）系统缺少足够的对换区空间：饭不会被修改的文件都直接从文件区调入；而当换出这些页面时，由于他们未被修改而不必再将它们换出。但对于那些可能被修改的部分，在将他们换出时需调到对换区，以后需要以后需要时再从对换区调入。

3）UNIX方式：与进程有关的文件都存放在文件去，故未运行过的页面都应从文件区调入。曾经运行过的但有被换出的页面，由于是被放在对换区，因此下次调入时应从对换区调入。进程请求的共享页面若被其他进程调入内粗你，则无需再从对换区调入。

5、抖动和工作集

在进程的页面置换过程中，频繁的页面调度行为成为抖动，或颠簸。如果一个进程在换页上用的时间多于执行时间，那么这个进程就在颠簸。

使用虚拟内存技术，操作系统中进程通常只有一部分块位于主存中，从而可以在内存中保留更多的进程以提高系统效率。此外，由于未用到的块不需要换入换出内存，因为节省了时间。但是系统必须很“聪明”地管理页面分配方案。在稳定状态，几乎主存的所有空间都被禁成块占据，处理器和操作系统可以直接访问到尽可能多的进程。但如果管理不当，系统发生抖动现象，处理器的大部分时间都将用于交换快，及请求调入页面的操作，而不是执行进程的指令，这就会大大降低系统效率。前面讲解的页面置换算法就是是讨论这里的分配方案，尽量避免抖动现象。

另外，为了防止出现抖动现象，需要选择合适的驻留集大小。驻留集（或工作集）是指在某段时间间隔内，进程要访问的页面集合。经常被使用的页面需要在驻留集中，而长期不被使用的页面要从驻留集中被丢弃。驻留集模型使用较为简单：操作系统跟踪每个进程的驻留集，并为进程分配大于驻留集的的空间。如果还有空闲，那么可启动另一个进程。如果所有驻留集之和增加一直超过了可用物理块啊的总数，那么系统会怎听一个进程，将其页面调出并且将其物理块分配给其他进程。

正确选择驻留集的大小，对存储器的有效利用和系统吞吐量的提高，都将产生重要的影响。

6、请求分段管理方式

请求分段存储管理系统已基本短时存储管理为基础，为用户提供一个比主存容量更大的虚拟存储器。作业的若干分段别放入内存，就可以开始作业运行，作业的其他部分被放在外存中，等到需要的时候才被调入内存。请求分段管理方式与请求分页存储管理方式类似，支不果断的大小不是固定的，在内存中空闲区域不够时，不能像分页一样采用简单的置换算法，可能需要置换多个端才有足够的空闲区。

1）段表机制。

在请求分段系统中，作业运行之前，只要求将当前需要的若干个分段装入内存，便可启动作业运行。在作业运行过程中，若要访问的分段不在内存，则通过缺段中断处理程序将其调入，同时还可以通过置换功能将暂时不用的分段调出到外存上，以便腾出与内存空间。为此，应对段表进程扩充。

在扩充后的段表项中，除了段号、段长、段在内存中的基址外，还增加了以下字段：

l 存取方式：表示本段的存取属性（只执行、只读或读写）。

l 访问字段A：记录该段被访问的频繁程度。

l 修改位M：表示该段进入主存后，是否已被修改，供置换段时参考。

l 存在位P：只是本段是否已调入主存，供程序访问时参考。

l 增补位：这是请求分段系统所特有的字段，表示本段在运行过程中是否有动态增长。

l 外存始址：只是本段在外村中的起始地址，即起始盘块号。

2）缺段中断机构

在请求分段系统中，当所要访问的段上未调入主存时，便由缺段中断机构产生一个缺段中断信号，请求操作系统将所要访问的段调入主存。

3）地址变换机构

请求分段系统中的地址变换机构在基本分段系统的地址变换机构的基础上，增加缺段中断请求及处理等形成。请求分段系统时以段为单位进行主存空间的分配，整段信息的装入、调出，有时需要主存空间的移动，这些都增加了系统的开销。

1）分段的共享

通过共享段来实现，每一个表项都是共享段的信息，记录了共享此段的每个进程情况。

2）分段的保护

越界检查：短号超过段表长度或段内偏移超过段长时做越界中断处理。

存取控制检查：段表项中存取控制字段规定了对该段的访问方式，比如只读、读写等。

环保护机构：一个程序可以访问驻留在相同环或较高特权环中的数据；一个程序可以调用驻留在相同环或较高特权环中的服务。

7、请求段页式管理方式

请求段页式管理方式只要求将作业若干页或段装入内存就可以开始运行作业，作业的其他部分别放在外存中，等待运行需要的时候才被调入内存，

请求段页式管理方式要求相对程序按逻辑意义分段后再分页，所以相对于请求页式管理方式能够方便用户使用，便于共享、保护和动态链接。进程在启动的时候采取与装入模式，则可以根据段的意义装入某些进程运行开始阶段所需要的段。

3.3、本章疑难点

分页管理方式和分段管理方式在很多地方相似，比如内存中都是不连续的、都有地址变换机构来进行地址映射。但两者也存在着许多区别。

1、目的

分页：页是信息的物理单位，分页是为了实现离散分配方式，以消减内存的外零头，提高内存的利用率。或者说，分页仅仅是由于系统管理的需要不是用户的需要。

分段：段是信息的逻辑单位，它含有一组其意义相对完整的信息，分段的目的是为了能更好的满足用户的需要。

2、长度

分页：页的大小固定且由系统决定，由系统把逻辑地址分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面，

分段：段的长度不固定，决定于用户所编写的程序，通常由编译程序在对流程序进行编译时，根据信息的性质来划分。

3、地址空间

分页：作业地址空间是一维的，即单一的线性地址空间，程序员只需利用一个级衣服，即可表示一个地址。

分段：作业地址空间是二维的，程序员在表示一个地址时，既需给出段名，又需给出段内地址。

4、碎片

分页：有内部碎片，无外部碎片。

分段：有外部碎片，无内部碎片。

5、共享和动态链接

分页：不容易实现

分段：容易实现

第四章 文件管理

4.1、文件系统基础

1、文件的概念

文件是操作系统中一个重要的概念。在系统运行时，计算机以进程为基本单位进行资源的调度和分配；而在用户进行的输入、输出中，则以文件为基本单位。大多数应用程序的输入都是通过文件来实现的，其输出也都保存在文件中，以便信息的长期存储及将来的访问。当用户将文件用于应用程序的输入、输出时，还希望可以访问文件、修改文件和保存文件等，实现对文件的维护管理，这就需要系统提供一个文件管理系统，操作系统的问价那系统就是实现用户的这些管理要求。

从用户的角度看，文件系统时操作系统的重要部分之一。用户关心的是如何命名、分类和查找文件，如何保证文件数据的安全性以及对文件可以进行哪些操作等。而对其中的细节，如文件如何存储在辅存上、如何管理晚间辅存区域等关心甚少。

文件系统提供了与二级存储相关的资源的映像，让用户能在不了解文件的各种属性、、文件存储介质的特性以及文件在存储介质上的具体位置等情况下，方便快捷的使用文件。

用户通过文件系统建立文件，提供应用程序的输入输出，对资源进行管理。首先了解文件的结构，我们通过自底向上的方式来定义。

1）数据项。

数据项是文件系统中最低级的数据组织形式，可分为以下两种类型：

基本数据项：用于描述一个对象的某种属性的一个值，如姓名、日期或证件号码等，是数据中可命名的最小逻辑数据单位，即原子数据。

组合数据项：有多个基本数据项组成。

2）记录。

记录是一组相关的数据项集合，用于描述一个对象在某方面的属性，如一个考生报名记录包括考生姓名、出生日期、报考学校代号、身份证号等一系列域。

3）文件。

文件是指由穿件这所定义的一组相关信息的集合，可分为有结构文件和无结构文件两种。在有结构文件中，文件由一组相似记录组成，如报考某学校的所有考生的报考信息记录；而无结构文件则被看成是一个字符流，比如一个二进制文件或字符文件。

虽然上面给出了结构化的表述，但实际上关于文件并无严格定义。通常在操作系统中将程序和数据组成文件。文件可以是数字、字母或二进制代码，基本访问单元可以是字节、行货记录。文件可以长期存储于硬盘或其他二级存储器，运行可控制的进程间共享访问，能够被组织成复杂的结构。

文件有一定的属性，这根据系统的不同而有所不同，但是通常都包括如下属性：

名称：文件名唯一，以容易读取的形势保存。

标示符：表示文件系统内文件的唯一标签，通常为数字，它是对人不可读的一种内部名称。

类型：被支持不同类型的文件系统所使用。

位置：指向设备和设备上文件的指针。

大小：文件当前大小（用字节、字或块表示），也可包含文件允许的最大值。

保护：对文件进行保护的访问控制信息。

时间、日期和用户标识：文件创建、上次修改和上次访问的相关信息，用于保护、安全和跟踪文件的使用。

所有文件的信息都保存在目录结构中，而目录结构也保存在外存上。文件信息当需要时再调入内存。通常，目录条目包括文件名称及其唯一标示符，而标示符定位其他属性的信息。

文件属于抽象数据类型。为了恰当的定义文件，就需要考虑有关文件的操作。操作系统提供系统调用，他对文件进行创建、写、读、定位和截断。

创建文件：创建文件有两个必要步骤。仪式在文件系统中为文件找到空间；而是在目录中为新文件创建条目。此目录条目记录文件名称、在文件系统中的位置以及其他可能的信息。

写文件：为了写文件，执行一个系统调用，指明文件名称和要写入文件的内容。对于给定文件名称，系统搜索目录以查找文件位置。系统必须为该文件维护一个写位置的指针。每当发生写操作，便更新写指针。

读文件：为了读文件，执行一个系统调用，指明文件名称和要读入文件块的内存位置。同样，需要搜索目录以找到相关目录项，系统维护一个读位置的指针。每当发生读操作时，更新读指针。一个进程通常只对一个文件读或写，所以当前操作位置可作为每个进程当前文件位置指针。由于读和写操作都是用同一指针，节省了空间也降低了系统复杂度。

文件重定位：（文件寻址）按某条件搜索目录，将当前文件位置设为给定值，并且不会读写文件。

删除文件：搜索到给定名称的文件并释放空间，找到相关目录并予以删除。

阶段文件：允许文件所有属性不变，并删除文件内容，即将其长度设为0并释放其空间。

这六个基本操作可以组成执行其他文件操作。例如，一个文件的复制，可以创建新文件；从旧文件读出并写入到新文件。

因为许多文件操作都涉及为给定文件搜索相关目录条目，许多系统要求在首次使用文件时，有系统调用open。操作系统维护一个包含所有打开文件信息的表（打开文件表，open-file table）。当需要一个文件操作时，可通过该表的一个索引指定文件，就省略了搜索环节。当

文件不再使用时，进程可以关闭它，操作系统从打开文件表中删除这一个条目。

大部分操作系统要求在文件使用之前就被显式的打开。操作open会根据文件名搜索目录，并将目录条目复制到打开文件表。如果调用open请求（创建、只读、读写、添加等）得到允许你，进程就可以打开文件，而open通常返回一个指向打开文件表中的一个条目的指针。通过使用该指针（而非文件名）进行所有IO操作，以简化步骤并节省资源。

整个系统表包含进程相关信息，如文件在磁盘的位置、访问日期和大小。一个进程打开一个文件，系统打开文件表就会为打开的文件增加一个条目，并指向整个系统表的相应条目。通常，系统打开文件表的每个文件时，还用一个文件打开计数器（open count），记录多少进程打开了该文件。每个关闭操作close则使count递减，当打开计数器为0时，便是该文件不再被使用。系统将收回分配给该文件的内存空间等资源，若文件被修改过，则将文件写会外存，并肩系统打开文件表中的相应条目删除，最后释放文件的文件控制块（file control block，FCB）。

每个打开文件都有如下关联信息：

l 文件指针：系统跟踪上次读写位置作为当前文件位置指针。，这种指针对打开文件的某个进程来说是唯一的，因此必须与磁盘文件属性分开保存。

l 文件打开计数：文件关闭时，操作系统必须重用其打开文件表条目，否则表内空间会不够用。因为多个进程可能打开同一个文件，所以系统在删除打开文件条目之前，必须等待最后一个进程关闭文件。该计数器跟踪打开或关闭的数量，当计数为0时，系统关闭文件，删除该条目。

l 文件磁盘位置：绝大多数文件操作都要求系统修改文件数据，该信息在内存中以免为每个操作都从磁盘中读取。

l 访问权限：每个进程打开文件都需要有一个访问模式（创建、只读、读写、添加等）。该信息保存在进程的打开文件表中一边操作系统能允许或拒绝之后的IO请求。

2、文件的逻辑结构

文件的逻辑结构是从用户观点出发看到的文件的组织形式。文件的物理结构是从省市县观点出发，又称为文件的存储结构，是指文件在外存上的存储组织形式。文件的逻辑结构与存储介质特征无关，但文件的物理结构与存储介质的特性有很大关系。

按逻辑结构，文件有无结构文件和有结构文件两种类型：

无机构稳健是最简单的文件组织形式。无结构文件将数据按顺序组织成记录并积累保存，它是有序相关信息项的集合，以字节（byte）为单位。由于无结构文件没有机构，因为对记录的访问只能通过穷举搜索的方式，股这种文件形势对大多数应用不适用，但字符流的无结构文件管理简单，用户可以方便地对其进行操作。所以，那些对基本信息单位操作不多的文件比较适于采用字符流的无结构方式，如源程序文件、目标代码文件等。

有结构文件按记录的组织形式可以分为：

1）顺序文件。

记录是定长的且按关键字顺序排列。可以顺序存储或以链表形势存储，在访问时需要顺序搜索文件。顺序文件有以下两种结构：

第一种是串结构，各记录之间的顺序与关键字无关。通常的办法是由时间来决定，即按存入时间的先后排列，最先存入的记录作为第一个记录，其次存入的为第二个记录，以此类推。

第二种是顺序结构，指文件中所有记录按关键字顺序排列。

在对记录进行批量操作时，即每次要读或写一大批记录，对顺序文件的效率是所有逻辑文件中最高的；此外，也只有顺序文件才能存储在磁带上，并能有效的工作。但顺序文件对查找、修改、增加或删除单个记录的操作比较困难。

2）索引文件

对于可变长记录的文件只能顺序查找，系统开销较大，为此可以建立一张索引表以加快检索速度，索引表本身是顺序文件。在记录很多或是访问要求高的文件中，需要引入索引以提供有效的访问，实际中，通过索引可以成百上千倍的提高访问速度。

3）索引顺序表

索引顺序表是顺序和索引两种组织形势的结合。索引顺序文件将顺序文件中所有记录分为若干个组，为顺序文件建立一张索引表，在索引表中为每组中的第一个记录建立一个索引项，其中含有该记录的关键字值和指向该记录的指针。

4）直接文件或散列文件（哈希文件，Hash File）

给定记录的键值或通过Hash函数转换的键值直接决定记录的物理地址。这种映射接哦股不同于顺序文件或索引文件，没有顺序的特性。散列文件有很高的存取速度，但是会引起冲突，即不同关键字的散列函数值相同。

3、目录结构

与文件管理系统和文件集合相关联的是文件目录，它包含有文件的信息，包括属性、位置和所有权等，这些信息都由操作系统进行管理。首先我们来看目录管理的基本要求 ：从用户的角度看，目录在用户所需要的文件名和文件之间提供一种映射，所以目录管理要实现“按名存取”；目录存取的效率直接影响到系统的性能，所以要提高对目录的检索速度；在共享系统中，目录还需要提供用于控制访问文件的信息。此外，文件允许重名也是用户的合理和必然要求，目录管理通过树形结构来解决和实现。

同进程管理一样，为实现目录管理，操作系统中引入了文件控制块的数据结构。

1）文件控制块。

文件控制块（FCB）是用来存放控制文件需要的各种信息的数据结构，以实现“按名存取”。FCB的有序集合称为文件目录，一个FCB就是一个文件目录项。为了创建一个新文件，系统将分配一个FCB并存放在文件目录中，称为目录项。

FCB主要包含以下信息：

l 基本信息，如文件名、文件的物理位置、文件的逻辑结构、文件的物理结构等。

l 存取控制信息，如文件的存取权限等。

l 使用信息，如文件建立时间、修改时间等。

2）索引节点

在检索目录文件的过程中，只用到了文件名，仅当找到一个目录项（查找文件名与目录项中文件名匹配）时，才需要从该目录项中独处该文件的物理地址。也就是说，在检索目录时，文件的其他描述信息不会用到，也不许调入内存，因此，有的系统（如UNIX）采用了文件名和文件描述信息分开的方法，文件描述信息单独形成一个称为索引节点的数据结构，简称为i节点。在文件目录中的每个目录项仅由文件名和指向该文件所对应的i节点的指针构成。

一个FCB的大小时64B，盘块大小是1KB，则在每个盘块中可以存放16个FCB。而在UNIX系统中一个目录仅占16B，其中14B是文件名，2B是i节点指针。在1KB的盘块中可存放64个目录项。这样可是查找文件时平均启动磁盘次数减少到原来的1/4，大大节省了系统开销。

存放在磁盘上的索引节点成为磁盘索引节点，UNIX中每个文件都有一个唯一的磁盘索引节点，主要包括以下几个方面：

文件主标示符，拥有该文件的个人或小组的标示符。

文件类型，包括普通文件、目录文件或特别文件。

文件存取权限，各类用户对该文件的存取权限。

文件物理地址，每个索引节点中含有13个地址项，即iaddr（0）~iaddr（12），他们以直接或间接方式给出数据文件所在盘块的编号。

文件长度，以字节为单位。

文件链接计数，文本文件系统中所有指向该文件的文件名的指针计数。

文件存取时间，本文件最近被进程存取的时间、最近被修改的时间以索引节点最近被修改的时间。

文件被打开时，磁盘索引节点复制到内存的索引节点中，以便于使用。在内存索引节点中有增加了以下内容：

索引节点编号，用于标示内存索引节点。

状态，指示i节点是否上锁或被修改。

访问计数，每当有一进程要访问此i节点时，计数加1，访问结束减1.

逻辑设备号，文件所属文件系统的逻辑设备号。

连接指针，设置分别指向空闲链表和散列队列的指针。

在理解一个文件系统的需求前，我们首先来了考虑在目录这个层次上所需要执行的操作，这有助于后面文件系统的整体理解。

搜索：当用户使用一个文件时，需要搜索目录，已找到该文件的对应目录项。

创建文件：当创建一个新文件时，需要在目录中增加一个目录项。

删除文件：当删除一个文件时，需要在目录中删除相应的目录项。

显示目录：用户可以请求显示目录的内容，如显示该用户目录中的所有文件及属性。

修改目录：某些文件属性保存在目录中，因而这些属性的变化需要改变相应的目录项。

操作时，考虑一下集中目录结构：

1）单级目录结构。

整个文件系统只建立一张目录表，每个文件占一个目录项。

当访问一个文件时，先按文件名在该目录中查找到相应的FCB，经合法性检查后执行相应的操作。当建立一个新文件时，必须先检索所有目录项以确保没有“重名”的现象，然后在该目录中增设一项，把FCB的全部信息保存在该项中。当删除一个文件时，先从该目录中找到该文件的目录项，回收该文件所占用的存储空间，然后再清除该目录项。

单级目录结构实现了按名存取，但是存在查找速度慢、文件不允许重名、不便于文件共享等缺点，而且对于多用户的操作系统显然是不适用的。

2）二级目录结构

单机目录很容易造成文件名称的混淆，可以考虑采用二级方案，将文件目录分成主文件目录MFD和用户文件目录UFD两级。

主文件目录项纪录用户名及相应用户文件所在的存储位置。用户文件目录项记录该用户文件的FCB信息。当某用户与对其文件进行访问时，只需要搜索该用户对应的UFD，这即解决了不同用户文件的重名问题，也在一定程度上保证了文件的安全。

两级目录结构可以解决多用户之间的文件重名问题，文件系统可以在目录上实现访问限制，但是两级结构对于用户结构内部结构没有任何帮助。用户如果需要在某个任务上进行合作和访问那其他文件时便会产生很多问题。

3）多级目录结构

也成为树形目录结构。将两级目录结构的层析加以推广，就形成了多级目录结构，及树形目录结构。

用户要访问某个文件时用文件的路径名标识文件，文件路径名是一个字符串，由从根目录出发到所找文件的通路商的所有目录名与数据文件名用分隔符/连接起来而成。从根目录出发的路径称为绝对路径。当层次较多时，每次从根目录查询浪费时间，于是加入了当前目录，进程对各文件的访问都是相对于当前目录进行的。当用户要访问某个文件时，使用相对路径标识文件，相对路径由从当前目录出发到所找文件通路商所有目录名与数据文件名用分隔符/链接而成。

通常，每个用户都有自己的当前目录，登陆后自动进入该用户的当前目录。操作系统提供一条专门的系统调用，供用户随时改变当前目录。

树形目录结构可以很方便的对文件进行分类，层次结构清晰，也能够更有效地进行文件的管理和保护。但是，在属性目录中查找一个文件，需要按路径名主机访问中间节点，这就增加了磁盘访问次数，无疑将影响查询速度。

4）五环图目录结构

树形目录结构可便于实现文件分类，但不便于实现文件共享，为此在树形目录结构的基础上增加了一些指向同一节点的有向边，使整个目录成为一个有向无环图。

引入五环图目录结构是为了实现文件共享。

当某用户要求删除一个共享节点时，若系统只是简单地将它删除，当另一共享用户需要访问时，却无法找到这个文件而发生错误。。为此可以为每个共享节点设置一个共享计数器，每当途中增加对该节点的共享链时，计数器加1；每当某用户提出删除该节点时，计数器减1。仅当共享计数器为0时，才眞正删除该节点，否则仅删除请求用户的共享链。

共享文件或目录不同于文件复制。如果有两个复制文件，每个程序员看到的是复制文件而不是原件；但如果一个文件被修改，那么另一个程序员的复制不会有改变。对于共享文件，只存在一个眞正文件，任何改变都会为其他用户所见。

无环图目录结构方便实现了文件的共享，但是的系统的管理变得更加复杂。

4、文件共享

文件共享十多个用户进程共享同一份文件，系统中只需保留该文件的一份副本。如果系统不能提供共享功能，那么每个需要该文件的用户都要有各自的副本，会造成对存储空间的极大浪费。

随着计算机技术的发展，文件共享的范围已由单机系统发展到多机系统，进而通过网络扩展到全球。这些文件的分享是通过分布式文件系统、远程文件系统、分布式信息系统实现的。这些系统允许多个客户通过c/s模型共享网络中的服务器文件。

现代常用的两种文件共享方法有：

在树形结构的目录中，当有两个或多个用户要共享一个子目录或文件时，必须将共享文件或子目录连接到两个或多个用户的目录中，才能方便的找到该文件。

在这种哦你共享方式中引用索引节点，即诸如文件的物理地址及其他的文件属性等信息，不再放在目录项中，而是放在索引节点中。在文件目录中只设置文件名及指向相应索引节点的指针。在索引节点中还应有一个连接技术count，用于表示连接到本索引节点上的用户目录项的数目。当count=2时，表示有两个用户目录项连接到本文件上，或者说是有两个用户共享此文件。

当用户A创建一个新文件时，他便是该文件的所有者，此时将count置1.当有用户B要共享此文件时，在用户B的目录中增加一个目录项，并设置一个指针指向该文件的索引节点，此时文件主仍然是用户A，count=2.如果用户A不再需要此文件，不能讲问价那直接删除。因为，若删除了该文件，也必然删除了该文件的索引节点，这样便会使用户B的指针悬空，而用户B则可能正在此文件上执行写操作，此时将因此半途而废。因而用户A不能拿删除此文件，只是将该文件count减1，然后删除自己目录中的相应目录项。用户B仍可以使用该文件。当count=0时，表示没有用户使用该文件，系统将负责删除该文件。

为使用户B能共享用户A的一个文件F，可以有系统创建一个link类型的新文件，也取名为F，并将文件F写入B的目录中，以实现用户B的目录与文件F的连接。在新文件中只包含被连接文件F的路径名。这样的连接方法被称为符号链接。

新文件中的路径名则被看作是符号链。当用户B要访问被链接的文件F且正要读LINK类新文件时，操作系统根据新文件中的路径名去读该文件，从而实现了用户B对文件F的共享。

在利用符号链方式实现文件共享时，只有文件的拥有者才拥有指向其索引节点的指针。而共享该文件的其他用户则只有该文件的路径名，并不拥有指向其索引节点的指针。这样，也就不会发生在文件主删除一个共享文件后留下一个悬空指针的情况。当文件的拥有者把一个共享文件删除后，其他用户通过符号链去访问它时，会出现访问失败，于是再将符号链删除，此时不会产生任何影响。此处我们要注意一个问题：当一个文件删除，而在使用其符号链接之前，另一个具有同样名称的文件被创建了。

在符号链的共享方式中，当其他用户读共享文件时，需要根据文件路径名逐个得查找目录，直至找到该文件的索引节点。因此，每次访问时，都可能要多次的读盘，使得访问文件的开销变大并增加了启动磁盘的频率。此外，符号链的索引结点也要耗费一定的磁盘空间。符号链方式有一个很大的优点，即网络共享只需提供该文件所在机器的网络地址以及该机器中的文件路径即可。

上述两种连接方式都存在一个共同的问题，即每个共享文件都有几个文件名。换言之，每增加一条链接，就增加一个文件名。这实质上就是每个用户都是用自己的路径名去访问共享文件。当我们试图去遍历整个文件系统时，将会多次遍历到该共享文件。

硬链接和软链接都是文件系统中的静态共享方法，在文件系统中还存在着另外的共享需求，及两个进程同时对同一个文件进行操作。这样的共享可以称为动态共享。

5、文件保护

为了防止文件共享可能会导致文件被破坏或未经核准的用户修改文件，文件系统必须控制用户对文件的存取，即解决对文件的读、写、执行的许可问题。为此，必须在文件系统中建立相应的文件保护机制。

文件保护通过口令保护、加密保护和访问控制等方式实现。其中，口令保护和加密保护是为了方式用户文件被他人存取或盗取，而访问控制则用于控制用户对文件的访问方式。

对文件的保护可以从限制对文件的访问类型中出发。可加以控制的访问类型主要有以下几种：

读：从文件中读。

写：向文件中写。

执行：将文件装入内存并执行。

添加：将信息添加到文件结尾部分。

删除：删除文件，释放空间。

列表清单：列出文件名和文件属性。

西外还可以对文件的重命名、复制、编辑等加以控制。这些该层的功能可以通过系统程序调用低层系统调用来实现。保护壳一直在底层提供。例如，复制文件可利用一系列的请求来完成。这样，具有读访问用户同时也具有复制和打印的权限了。

解决访问控制最常用的方法是根据用户身份进行控制。而实现基于身份访问的最为普通的方法是为每个文件和目录增加一个访问控制列表（Access-Control List，ACL），以规定每个用户名及其所允许访问的类型。

这种方法的优点是可以使用复杂的访问方法。其缺点是长度无法预期并且可能导致复杂的空间管理，使用精简的访问列表可以解决这个问题。

精简的访问列表采用拥有者、组合其他三种用户类型。

1）拥有者：创建文件的用户。

2）组：一组需要共享文件且具有类似访问的用户。

3）其他：系统内的所有其他用户。

这样只需用三个域列出访问表中这三类用户的访问权相即可。文件拥有者在创建文件时，说明创建者用户名及所在的组名，系统在创建文件时也将文件主的名字、所属组名列在该文件的FCB中。用户访问该文件时，按照拥有着所拥有的权限访问。UNIX操作系统即采用此种方法。

口令和密码是另外两种访问控制方法。

口令指用户在建立一个文件时提供一个口令，系统为其建立FCB时附上相应的口令，同时告诉允许共享该文件的其他用户。用户请求访问时必须提供相应的口令。这种方法时间和空间的开销不多，缺点是口令直接存在系统内部，不够安全。

密码纸用户对文件进行加密，文件被访问时需要使用密钥。这种方法保密性强，节省了存储空间，不过编码和译码要花费一定时间。

口令和密码都是仿制用户文件被他人存取或盗取，并没有控制用户对文件的访问类型。

注意两个问题：

1）现代操作系统常用的文件保护方法，是将访问控制列表与用户、组和其他成员访问控制方案一起组合使用。

2）对于多级目录结构而言，不仅需要保护单个文件，而且还需要保护子目录内的文件，即需要提供目录保护机制。目录操作与文件操作并不相同，因此需要不同的保护机制。

4.2、文件系统实现

1、文件系统层次结构

现代操作系统有多种文件系统类型，因此文件系统的层次结构也不尽相同。

文件系统为用户提供与文件及目录有关的调用，如新建、打开、读写、关闭、删除文件，建立、删除目录等。此层由若干程序模块组成，每一模块对应一条系统调用，用户发出系统调用时，控制即转入相应的模块。

文件目录系统的主要功能是管理文件目录，其任务有管理活跃文件目录表、管理读写状态信息表、管理用户进程的打开文件表、管理与组织在存储设备上的文件目录结构、调用下一级存取控制模块。

实现文件保护主要由该级软件完成，它把用户的访问要求与FCB中指示的访问控制权限进行比较，以确认访问的合法性。

逻辑文件系统与文件信息缓冲区的主要功能是根据文件的逻辑结构将用户要读写的逻辑记录转换成文件的逻辑结构内的相应块号。

物理文件系统的主要功能是把逻辑记录所在的相对块号转换成实际的物理地址。

分配模块的主要功能是管理辅存空间，即2负责分配辅存空闲空间和回收辅存空间。

设备管理程序模块的主要功能是分配设备、分配设备读写缓冲区、磁盘调度、启动设备、处理设备中断、释放设备读写缓冲区、释放设备等。

2、目录实现

在读文件前，必须先打开文件。打开文件时，操作系统利用路径名吵到相应目录项，目录项中提供了查找文件磁盘块所需要的信息，目录实现的基本方法有线性列表和哈希表两种方法。

最简单的目录实现方法是使用存储文件名和数据块指针的线性表。创建新文件时，必须首先搜索目录表以确定没有同名的文件存在，然后在目录表后增加一个目录项。删除文件则根据给定的文件名搜索目录表，接着释放分配给他的空间。若要重用目录项，有许多方法：可以将目录项标记为不再使用，或者将它加到空闲目录项表上，还可以将目录表中最后一个目录项复制到空闲位置，不过由于线性表的特殊性，运行比较费时。

哈希表根据文件名得到一个值，并返回一个指向线性列表中元素的指针。这种方法的有点事查找非常迅速，插入和删除也较简单，不过需要一些预备措施来避免冲突。最大的困难是哈希表长度固定以及哈希函数对表长的依赖性。

目录查询必须通过在磁盘上反复搜索完成，需要不断的进行IO操作，开销较大。所以如前面所述，为了减少IO操作，把当前使用的文件目录复制到内存，以后要使用该文件时只要在内存中操作，从而见底了磁盘操作次数。

3、文件实现

文件分配对应于文件的物理结构，是指如何为文件分配磁盘块。常用的磁盘空间分配方式有三种：连续分配、链接分配和索引分配。有的系统对三种方式都支持，但是更普遍的是一个系统只提供一种方法支持。

1）连续分配。

连续分配方法要求每个文件在磁盘上占有一组连续的块。磁盘地址定义了磁盘上的一个线性排序。这种排序使作业访问磁盘时需要的寻道数和寻道时间最小。

文件的连续分配可以用第一块的磁盘地址和连续块的数量来定义。如果文件有n块长并从位置b开始，那么该文件将占有b，b+1，b+2……b+n-1.一个文件的目录条目包括开始块的地址和该文件所分配区域的长度。

连续分配支持顺序访问和直接访问。其优点是实现简单、存取速度快。缺点在于，文件长度不宜动态增加，因为一个文件末尾后的盘块可能已经分配给其他文件，一旦需要增加，就需要大量移动盘块。此外，反复增删文件后会产生外部碎片（与内存管理分配方式中的碎片相似），并且很难确定一个文件需要的空间大小，因而只适用于长度固定的文件。

2）链接分配

链接分配解决了连续分配的碎片和文件大小问题。采用链接分配，每个文件对应一个磁盘块的链表；磁盘块分布在磁盘的任何地方，除最后一个盘块外，每个盘块都有指向下一个盘块的指针，这些指针对用户是透明的。目录包括文件第一块的指针和最后一块的指针。

创建新文件时，目录中增加一个新条目。链接分配中每个目录项都有一个指向文件首块的指针。该指针初始化为nil以表示空文件，大小字段为0.写文件会通过空闲空间管理系统找到空闲块，将该块链接到文件的尾部，以便于写入。读文件则通过块到块的指针读块。

链接分配方式没有外部碎片，空闲空间列表上的任何块都可以用来满足请求。创建文件时并不需要说明文件大小。只要有空闲块文件就可以增大，也无需合并磁盘空间。

链接分配的缺点在于无法直接访问盘块，只能通过指针顺序访问文件，以及盘块指针消耗了一定的存储空间。链接分配方式的稳定性也是一个问题。

系统在运行过程中由于软件或者硬件错误导致链表中的指针丢失或损坏，会导致文件数据的丢失。

3）索引分配

连接分配解决了连续分配的外部碎片和文件大小管理的问题。但是，链接分配不能有效支持直接访问（FAT除外）。索引分配解决了这个问题，他把每个文件的所有的盘块号都集中在一起构成索引块。

每个文件都有其索引块，这是一个磁盘块地址的数组。索引块的第i个条目指向文件的第i个块。目录条目包括索引块的地址。要读第i块，通过索引块的第i个条目的指针来查找和读入所需的块。

创建文件时，索引块的所有指针都设为空。当首次写入第i块时，先从空闲空间中取得一个块，再将其地址写到索引块的第i个条目。索引分配支持直接访问，且没有外部碎片问题。其缺点是由于索引块的分配，增加了系统存储空间的开销。索引块的大小是一个重要的问题，每个文件必须有一个索引块，因此索引块应尽可能小，但索引块太小就无法支持大文件。可以采用以下机制来处理这个问题。

链接方案：一个索引块通常为一个磁盘块，因此，他本身能直接读写。为了处理大文件，可以将多个索引块链接起来。

多层索引：多层索引使第一层索引块指向第二层的索引块，第二层的索引块再指向文件块。这种方法根据最大文件的大小的要求，可以继续到第三层或第四层。例如，4096B的块，能在索引块中存入1024个4B的指针。两层索引允许1048576个数据块，即允许最大文件为4G。

混合索引：将多种索引分配方式相结合的分配方式。例如，系统即采用直接地址，又采用单级索引分配方式或两级索引分配方式。

此外，访问文件需要两次访问外存——手想要读取索引块的内容，然后在访问具体的磁盘块，因而降低了文件的存取速度。为了解决这一问题，通常将文件的索引块读入内存的缓冲区，以加快文件的访问速度。

1）文件存储管理空间的划分与初始化。

一般来说，一个文件存储在一个文件卷中。文件卷可以是物理盘的一部分，也可以是整个物理盘，支持超大型文件的文件卷也可以是有多个物理盘组成。

在一个文件卷中，文件数据信息的空间（文件区）和存放文件控制信息FCB的空间（目录区）是分离的。由于存在很多种类的文件表示和存放格式，所以现代操作系统中一般都有很多不同的文件管理模块，通过他们可以访问不同格式的逻辑卷中的文件。逻辑卷在提供文件服务前，必须由对应的文件程序进行初始化，划分好目录区和文件区，建立空闲空间管理表格及存放逻辑卷信息的超级块。

2）文件存储器空间管理

文件存储设备分成许多大小相同的物理块，并以块为单位交换信息，因此，文件存储设备的管理实质上是对空闲块的组织和管理，它包括空闲块的组织，分配与回收等问题。

l 空闲表法。

空闲表法属于连续分配方式，它与内存的动态分配方式类似，为每个文件分配一块连续的存储空间。系统为外存上的所有空闲区建立一张空闲盘块表，每个空闲区对应一个空闲表项，其中包括表项序号、该空闲区第一个盘块号、该区的空闲盘块数等信息。再将所有空闲区按其起始盘块号递增的次序排列。

空闲盘区的分配与内存的动态分配类似，同样是采用首次适应算法、循环首次适应算法等。例如，在系统为某新创建的文件分配空闲盘块时，先顺序的检索空闲盘块表的各表项，直至找到第一个其大小能满足要求的空闲区，再将该盘区分配给用户（进程），同时修改空闲盘块表。

系统在对用户所释放的存储空间进行回收时，也采取类似于内存回收的方法，即要考虑回收区是否与空闲表中插入点的前区和后区相邻接，对相邻接者应予以合并。

| 空闲链表法

将所有空闲盘区拉成一条空闲链，根据构成连所有的基本元素不同，可以把链表分成两种形式：空闲盘块链和空闲盘区链。

空闲盘块链是将磁盘上所有空闲空间，以盘块为单位拉成一条链。当用户因创建文件而请求分配存储空间时，系统从链首开始，一次摘下适当数目的空闲盘块分配给用户。当用户因删除文件而释放存储空间时，系统将回收的盘块一次拆入空闲盘块链的末尾。这种方法的优点是分配和回收一个盘块很简单，但在为一个文件分配盘块时，可能要重复多次操作。

空闲盘区链是将磁盘上所有空闲盘区（每个盘区可包含若干个盘块）拉成一条链。在每个盘区上除含有用于指示下一个空闲盘区的指针外，还应有能指明本盘区大小（盘块数）的信息。分配盘区的方法与内存的动态分区分配类似，通常采用首次适应算法。在回收盘区时，同样也要将回收区域相邻接的空闲盘区相合并。

| 位视图法。

位视图是利用二进制的以为来表示磁盘中的一个盘块的使用情况，磁盘上所有的盘块都有一个二进制位与之对应，当其直为0时，表示对应的盘块空闲；当其值为1时，表示对应的盘块已分配。

盘块的分配：

1顺序扫描位视图，从中找出一个或一组其值为0的二进制位。

2将所找到的一个或一组二进制位，转换成与之对应的盘块号。假定找到的其值为0的二进制位，位于位视图的第i行，第j列，则其相应的盘块号硬干下式计算：b=n（i-1）+j

3修改位视图，令map【i，j】=1

盘块的回收：

1将回收盘块的盘块号转换成位视图中的行号和列号

2修改位视图，令map【i，j】=0

I 成组链接法。

空闲表法和空闲链表法都不适用于大型文件系统，因为这回事空闲表或空闲链表太大。在UNIX系统中采用的是成组链接法，这种发发结合了空闲表和空闲链表法两种方法，克服了表太大的缺点。其大致思想是：把顺序的n个空闲扇区地址保存咋i第一个空闲扇区内，其后一个空闲扇区则保存另一顺序空闲扇区的地址，如此继续直至所有空闲扇区均予以链接。系统只需要保存一个指向第一个空闲扇区的指针。

表示文件存储器空闲空间的位向量表或第一个成组链块以及卷中的目录区、文件区划分都需要存放在辅存储器-中的，一般放在卷头位置，在UNIX系统中成为超级块。在对卷中文件进行操作前，超级块需要预先读入系统空间的主存，并且经常保持主存超级块与辅存卷中超级块的一致性。

4.3、磁盘组织与管理

1、磁盘的结构

磁盘是由表面涂有磁性物质的金属或塑料构成的圆形盘片，通过一个称谓磁头的到体系安全从磁盘中存取数据。在读写操作期间，磁头固定，磁盘在下面高速旋转。磁盘的表面上数据存储在一组同心圆中，称为磁道。每个磁道与磁头一样快，一个盘面上有上千个磁道。磁道有划分为几百个扇区，每个扇区固定存储大小（通常为512B），一个扇区称为一个盘块。由于扇区按固定圆心角划分，所以每季度从最外道向里道增加，磁盘的存储能力受限于最内道的最大记录密度。

磁盘地址用“柱面号-盘面号-扇区号（或块号）”来表示。

2、磁盘调度算法

目前常用的磁盘调度算法有以下几种：

1）先来先服务（FCFS）算法

FCFS算法根据进程请求访问磁盘的先后顺序进行调度处理，这是一种最简单的调度算法。这种算法的优点是具有公平性。如果只有少量进程需要访问，且大部分请求都是访问簇聚的文件扇区，则会达到较好的性能；但如果有大量进程竞争使用磁盘，那么这种算法在性能上往往低于随即调度。所以，实际磁盘调度中考虑一些更为复杂的调度算法。

2）最短寻找时间优先（SSTF）算法

SSTF选择调度处理的磁道是与当前磁头所在磁道距离最近的磁道，一是每次的寻找时间最短。当然，总是选择最小寻找时间并不能保证平均寻找时间最小，但是能提供比FCFS算法更好的性能。这种算法会产生饥饿现象。

3）扫描（SCAN）算法又称为电梯算法

SCAN算法在磁头当前移动方向上选择与当前磁头所在磁道距离最近的请求作为下一次服务的对象。由于磁头移动规律与电梯运行相似，故又称为电梯调度算法。SCAN算法对最扫描过的区域不公平，因此，他在访问局部性方面不如FCFS算法和SSTF算法好。

4）循环扫描（C-SCAN）算法

在扫面算法的基础上规定磁头单向移动来提供服务，回返时直接快速移动至起始端而不服务任何请求。由于SCAN算法偏向于处理那些接近最里或最外的磁道的访问请求，所以使用改进型的C-SCAN算法来避免这个问题。

采用scan算法和c-scan算法时磁头总是严格地遵循从盘面的一端到另一端，显然在实际使用时还可以改进，即磁头移动只需要到达最远端的一个请求即可返回，不需要到达磁盘端点这种形式的SCAN算法和C-SCAN算法成为LOOK和C-LOOK调度。这是因为它们在朝一个给定方向移动前会查看是否有请求。

对比以上几种磁盘调度算法，FCFS算法太简单，性能较差，仅在请求队列长度接近于1时才较为理想；SSTF算法较为通用和自然；SCAN算法和C-SCAN算法在磁盘负载较大时比较占优势。

除了减少寻找时间外，减少延迟时间也是提高磁盘传输效率的重要因素。可以对盘面扇区进行交替编号，对磁盘片组中的不同盘面错位命名。

磁盘时连续自转设备，磁盘机读写一个物理块后，需要经过短暂的处理时间才能开始读写下一个块。假设逻辑记录数据连续存放在磁盘空间中，若在盘面上按扇区交替编号连续存放，则连续读写多个记录时能够减少磁头的延迟时间；同柱面不同扇面的扇区若能错位编号，连续读写相邻两个盘面的逻辑记录时也能减少磁头延迟时间。

3、磁盘的管理

一个新的磁盘只是一个含有磁性记录材料的空白盘。在磁盘能存储数据之前，它必须分成扇区以便磁盘控制器能进行读和写的操作，这个过程称为低级格式化（物理分区）。低级格式化为磁盘的每个扇区采用特别的数据结构。每个扇区的数据结构通常由头、数据区域（通常为512B）大小和尾部组成。头部和尾部包含了一些磁盘控制器所使用的信息。

为了使用磁盘存储文件。操作系统还需要将自己的数据结构记录在磁盘上：第一步将磁盘分为一个或多个柱面组成分区；低而不对物理分区进行逻辑格式化，操作系统将出师的文件系统数据结构存储在磁盘上，这些数据结构包括空闲和已分配的空间以及一个初始为空的目录。

（2）引导块

计算机启动时需要运行一个初始化程序（自举程序），它初始化CPU、寄存器、设备控制器和内存等，接着启动操作系统。为此，该自举程序应找到磁盘上的操作系统内核，装入内存，并转到起始地址，从而开始操作系统的运行。

自举程序通常保存在ROM中，为了避免改变自举代码需要改变ROM硬件的问题，故指在ROM中保留很小的自举装入程序，将完整功能的自举程序保存在磁盘的启动块上，启动块位于磁盘的固定位。拥有启动分区的磁盘称为启动磁盘或者系统磁盘。

（3）坏块

由于磁盘有移动部件且容错能力弱，所以容易导致一个或多个扇区损坏。部分磁盘甚至从出厂时就有坏扇区。根据所使用的磁盘和控制器，对这些坏块有多种处理方式。

对于简单磁盘，坏扇可手工处理。

对于复杂的磁盘，其控制器维护一个磁盘坏块链表。该链表在出厂前进行低级格式化就初始化了，并在磁盘的整个使用过程中不断更新。低级格式化将一些块保留作为备用，对操作系统透明。控制器可以使用备用块来逻辑地代替坏块，这种方案称为扇区备用。

4.4、本章疑难点

1、磁盘结构

引导控制块（Boot Control Block）包括系统从该分区引导操作系统所需要的信息。如果磁盘没有操作系统，那么这块内容为空。它通常为分区的第一块。UFS称之为引导块；NTFS称之为分区引导扇区。

分区控制块包括分区详细信息，如分区的块数、块的大小、空闲块的数量和指针、空闲FCB的数量和指针等。UFS称之为超级块；而NTFS称之为主控文件表。

2、内存结构

内存分区表包含所有安装分区的信息。

内存目录结构用来保存近来访问过的目录信息。对安装分区的目录，可以包括一个指向分区表的指针。

系统范围的打开文件表，包括每个打开文件的FCB复制和其他信息。

单个进程的打开文件表，包括一个指向系统范围内已打开文件表中适合条目和其他信息的指针。

3、文件系统实现概述

为了创建一个文件，应用程序调用逻辑文件系统。逻辑文件系统知道目录结构形势，它将分配一个新的FCB给文件，把相应目录读入内存，用新的文件名更新该目录和FCB，并将结果写回到磁盘。

一旦文件被创建，它就能用于IO，不过首先要打开文件。调用open将文件名传给文件系统，文件系统根据给定文件名搜索目录结构。部分目录结构通常缓存在内存中以加快目录操作。找到文件后，其FCB复制到系统范围的打开文件表。该表不但存储FCB，也有打开该文件的进程数量的条目。

然后，单个进程的打开文件表中会增加一个条目，并通过指针将系统范围的打开文件表的条目同其他域（文件当前位置的指针和文件打开模式等）相连。调用open返回的是一个指向单个进程的打开文件表中合适条目的指针。所以文件操作都是通过该指针进行。

文件名不必是打开文件表的一部分，因为一旦完成对FCB在磁盘上的定位，系统就不再使用文件名了。对于访问打开文件表的索引，UNIX称之为文件描述符；而Windows称之为文件句柄。因此，只要文件没有被关闭，所有文件操作通过打开文件表来进行。

当一个进程关闭文件，就删除一个相应的单个进程打开文件表的条目，系统范围内打开文件表的打开数也会递减。当打开文件的所有用户都关闭了一个文件时，更新的文件信息会复制到磁盘的目录结构中，系统范围打开的文件表的条目也将删除。

在实际中，系统调用open会首先搜索系统范围的打开文件表以确定某文件是否已被其他进程使用。如果是，就在单个进程的打开文件表中创建一项，并指向现有系统范围的打开文件表的相应条目。该算法在文件已打开时，能节省大量开销。

4、混合索引分配的实现

混合索引分配已在UNIX系统中采用。在UNIX System V的索引结点中，共设置了13个地址项，即iaddr（0）~iaddr（12）。在BSD UNIX的索引结点中，共设置了13个地址项，它们把所有的地址项分成两类，即直接地址和间接地址。

（1）直接地址

为了提高对文件的检索速度，在索引结点中可设置10个直接地址项，即用iaddr（0）~iaddr（9）来存放直接地址。换言之，在这里的每项中所存放的是该文件数据所在盘块的盘块号。假如每个盘块的大小为4KB，当文件不大于40KB时，便可直接从索引节点中读出该文件的全部盘块号。

（2）一次间接地址

对于大、中型文件，只采用直接地址并不现实。可再利用索引节点中的地址项iaddr（10）来提供一次间接地址。这种方式的实质就是一级索引分配方式。一次间址块就是索引块，系统将分配给文件的多个盘块号记入其中。在一次间址块中可存放1024个盘块号，因而允许文件长达4MB。

（3）多次间接地址

当文件长度大于4MB+40KB时，系统还须采用二次间址分配方式。这是用地址项iaddr（11）提供二次间接地址。该方式的实质是二级索引分配方式。系统此时是在二次间址块中记入所有一次间址块的盘号。在采用二次间址方式时，文件最大长度可达4GB。同理，地址项iaddr（12）作为三次间接地址，其所允许的文件最大长度可达4TB。

第五章 输入输出管理

5.1、IO管理概述

1、IO设备

IO设备管理是操作系统设计中最凌乱也最具挑战性的部分。由于它包含了很多领域的不同设备以及与设备相关的应用程序，因此很难有一个通用且一直的设计方案。所以在理解设备管理之前，应该先了解具体的IO设备类型。

计算机系统中的IO设备按使用特性可以分为一下类型：

1）人机交互类外部设备，又称慢速IO设备，用于桶计算机用户之间交互的设备，如打印机、显示器、鼠标、键盘等。这类设备数据交换速度相对较慢，通常是以字节为单位进行数据交换。

2）存储设备，用于存储程序和数据的设备，如磁盘、磁带、光盘等。这类设备用于数据交换，速度较快，通常以多字节组成的块为单位进行数据交换。

3）网络通信设备，用于与远程设备通信的设备，如各种网络接口、调制解调器等。其数据交换速度介于外部设备与存储设备之间。网络通信设备在使用和管理上与前两者设备有很大的不同。

1）低速设备，传输速率仅为每秒钟几个字节至数百个字节的一类设备，如键盘、鼠标等。

2）中速设备，传输速率在每秒数千个字节至数万个字节的一类设备，如行式打印机、激光打印机等。

3）高速设备，传输速率在数百个千字节至千兆字节的一类设备，如磁带机、磁盘机、光盘机等。

（2）按信息交换的单位分类

1）块设备

由于信息的存取总是以数据块为单位，所以存储信息的设备称为块设备。它属于有结构设备，如磁盘等。磁盘设备的基本特征是传输速率高，以及可寻址，即对他可随机地读写任意块。

2）字符设备

用于数据输入输出的设备为字符设备，因为其传输的基本单位是字符。它属于无结构类型，如交互式终端机、打印机等。他们的传输速率低、不可寻址、并且在输入输出时常采用中断驱动方式。

对于IO设备，有以下三种不同类型的使用方式：

独占式使用设备。独占式使用设备是指在申请设备是，如果设备空闲，就将其独占，不再允许其他进程申请使用，一直等到该设备被释放才允许其他进程申请使用。例如：打印机。

分时式共享使用设备。独占式使用设备时，设备利用率低，当设备没有独占使用的要求时，可以通过分时共享使用，提高利用率。例如：对磁盘设备的IO操作，各进程每次IO操作请求可以通过分时来交替进行。

以SPOOLing方式使用外部设备。SPOOLing技术是在批处理操作系统时代引入的，即假脱机IO技术。这种技术用于对设备的操作，实质上就是对IO操作进行批处理。具体的内容后面有单独讲解。

采用上面三种使用方式的设备分别称为独占设备、共享设备和虚拟设备。

2、IO管理目标

IO设备管理的主要目标有以下三个方面。

方便使用：方便用户使用外部设备，控制设备工作完成用户的输入输出要求。

提高效率：提高系统的并行工作能力，提高设备的使用效率。

方便控制：提高外围设备和系统的可靠性和安全性，以使系统能正常工作。

3、IO管理功能

IO设备管理的功能是按照输入输出子系统的结构和设备类型制定分配和使用设备的策略，主要包括：

设备的分配和回收。

外围设备的启动。

对磁盘的驱动调度。

外部设备中断处理。

虚拟设备的实现。

4、IO应用接口

IO应用接口就是从不同的输入输出设备中抽象出一些通用类型。每个类型都可以通过一组标准函数（即接口）来访问。具体的差别被内核模块（也称设备驱动程序）所封装。这些设备驱动程序一方面可以定制，以设和各种设备，另一方面也提供了一些标准接口。

IO应用接口的具体实现方式是：先把IO设备划分为若干种类的通用类型；然后对每一种类型提供一组标准函数来访问，这里的标准函数就是接口；为每个IO设备提供各自的设备驱动程序，各种设备间的差异就体现在设备驱动程序的不同之中，而对于访问这些设备的接口却是按照该设备分数的类型而统一。

划分IO设备所属的通用类型的依据：

l 字符设备还是块设备。

l 顺序访问还是随机访问。

l IO传输是同步还是异步。

l 共享设备还是独占设备。

l 操作速度的高低。

l 访问模式是读写、只读还是只写。

5、设备控制器（IO部件）

IO设备通常包括一个机械部件和一个电子部件。为了达到设计的模块性和通用性，一般将其分开。电子部件成为设备控制器（或适配器），在个人计算机中，通常是一块插入主板扩充槽的印制电路板；机械部件即设备本身。

由于具体的设备操作涉及硬件接口，且不同的设备有不同的硬件特性和参数，所以这些复杂的操作交由操作系统用户编写程序来操作是不实际的。引入控制器后，系统可以通过几个简单的参数完成对控制器的操作，而具体的硬件操作则由控制器调用相应的设备接口完成。设备控制器的引入大大简化了操作系统的设计，特别是有利于计算机系统和操作系统对各类控制器和设备的兼容；同时也实现了主存和设备之间的数据传输操作，使CPU从繁重的设备控制操作中解放出来。

设备控制器通过寄存器与CPU通信，在某些计算机上，这些寄存器占用内存地址的一部分，称为内存映像IO；另一些计算机则采用IO专用地址，寄存器独立编址。操作系统通过想控制器寄存器写命令字来执行IO功能。控制器收到一条命令后，CPU可以转向进行其他工作，而让设备控制器自行完成具体IO操作。当命令执行完毕后，控制器发出一个中断信号，操作系统重新获得CPU的控制权并检查执行结果，此时，CPU仍旧是从控制器寄存器中读取信息来获得执行结果和设备的状态信息。

设备控制器的主要功能为：

l 接收和识别CPU或通道发来的命令，如磁盘控制器能就收读、写、查找、搜索等命令。

l 实现数据交换，包括设备和控制器之间的数据传输；通过数据总线或通道，控制器和主存之间的数据传输。

l 发现和记录设备及自身的状态信息，供CPU处理使用。

l 设备地址识别。

为实现上述功能，设备控制器必须包含以下组成部分：

该接口有三类信号线：数据线、地址线和控制线。数据线通常与两类寄存器相连接：数据存储器（存放从设备送来的输入数据或从CPU送来的输出数据）和控制/状态寄存器（存放从CPU送来的控制信息或设备的状态信息）。

设备控制器链接设备需要相应数量的接口，一个借口链接一台设备。每个接口中都存在数据、控制和状态三种类型的信号。

用于实现对设备的控制。它通过一组控制线与处理器交互，对从处理器收到的IO命令进行译码。CPU启动设备时，将启动命令发送给控制器，并同时通过地址线吧地址发送给控制器，由控制器的IO逻辑对地址进行译码，并相应地对所选设备进行控制。

6、IO控制方式

设备管理的主要任务之一是控制设备和内存或处理器之间的数据传送，外围设备和内存之间的输入输出控制方式有四种，下面分别介绍。

计算机从外部设备读取数据到存储器，每次读一个字的数据。对读入的每个字，CPU需要对状态循环检查，知道确定该字已经在IO控制器的数据寄存器中。在程序IO方式中，由于CPU的高速型和IO设备的低速性，致使CPU的绝大部分时间都处于等待IO设备完成数据IO的循环测试中，造成CPU的极大浪费。在该方式中，CPU之所以要不断地测试IO设备的状态，就是因为在CPU中无中断机构，使IO设备无法向CPU报告它已完成了一个字符的输入操作。

程序直接控制方式虽然简单易于实现，但是其缺点也是显然的，由于CPU和外部设备只能串行工作，导致CPU的利用率相当低。

中断驱动方式的思想是：允许IO设备主动打断CPU的运行并请求服务，从而“解放”CPU，使得其向IO控制器发送命令后可以继续做其他有用的工作。我们从IO控制器和CPU两个角度分别来看中断驱动方式的工作过程： 从IO控制器的角度来看，IO控制器从COU接受一个读命令，然后从外围设备读数据。一旦数据读入到该IO控制器的数据寄存器，便通过控制线给CPU发出一个中断信号，表示数据已准备好，然后等待CPU请求该数据。IO控制器收到CPU发出的取数据请求后，将数据放到数据总线上，传到CPU的寄存器中。至此，本次IO操作完成，IO控制器又可以开始下一次IO操作。

从CPU的角度来看，CPU发送读命令，然后保存当前运行程序的上下文（现场，包括程序计数器及处理器寄存器），转去执行其他程序。在每个指令周期的末尾，CPU检查中断。当有来自IO控制器的中断时，CPU保存当前正在运行程序的上下文，转去执行中断处理程序处理该中断。这时，CPU从IO控制器读一个字的数据传送到寄存器，并存入主存。接着，CPU恢复发出IO命令的程序（或其他程序）的上下文，然后继续运行。

中断驱动方式比程序直接控制方式有效，但由于数据中的每个字在存储器与IO控制器之间的传输都必须通过CPU处理，这就导致了中断驱动方式仍然会花费较多的CPU时间。

中断驱动方式中，CPU仍然需要主动处理在存储器和IO设备之间的数据传送，所以速度还是受限，而直接内存存取（DMA）方式的基本思想是在外围设备和内存之间开辟直接的数据交换通路，彻底解放CPU。该方式的特点是：

l 基本单位是数据块。

l 所传诵的数据，是从设备直接送入内存的，或者相反。

l 仅在传送一个或多个数据块的开始和结束时，才需CPU干预，整块数据的传送是在DMA控制器的控制下完成的。

为了实现在主机与控制器之间成块数据的直接交换，必须在DMA控制器中设置如下四类寄存器：

l 命令/状态寄存器（CR）。用于接收从CPU发来的IO命令或有关控制信息，或设备的状态。

l 内存地址寄存器（MAR）。在输入时，它存放把数据从设备传送到内存的起始目标地址；在输出时，它存放由内存到设备的内存源地址。

l 数据寄存器（DR）。用于暂存从设备到内存或从内存到设备的数据。

l 数据计数器（DC）。存放本次CPU要读或写的字节数。

DMA的工作过程是：CPU读写数据时，他给IO控制器发出一条命令，启动DMA控制器，然后继续其他工作。之后CPU就把这个操作委托给DMA控制器，由该控制器负责处理。DMA控制器直接与存储器交互，传送整个数据块，这个过程不需要CPU参与。当传送完成后，DMA控制器发送一个中断信号给处理器。因此，只有在传送开始和结束时才需要CPU的参与。

DMA控制方式与中断驱动方式的主要区别是中断驱动方式在每个数据传送玩后中断CPU，而DMA控制方式则是在所要求传送的一批数据全部传送结束时中断CPU；此外，中断驱动方式数据传送的是在中断处理时由CPU控制完成，而DMA控制方式则是在DMA控制器的控制下完成的。

IO通道方式是DMA方式的发展，它可以进一步减少CPU的干预，即把对一个数据块的读或写为一个单位的干预，减少为对一组数据块的读或写及有关的控制盒管理为单位的干预。同时，又可以实现CPU、通道和IO设备三者的并行操作，从而更有效的提高整个系统的资源利用率。

例如，当CPU要完成一组相关的读或写操作及有关控制时，只需向IO通道发送一条IO指令，已给出其所要执行的通道程序的首址和要访问的IO设备，通道接到该指令后，通过执行通道程序便可完成CPU指定的IO任务。

IO通道和一般处理器的区别是：通道指令的类型单一，没有自己的内存，通道所执行的通道程序释放在主机内存中的，也就是说通道与CPU共享内存。

IO通道与DMA的区别是：DMA方式需要CPU来控制传输的数据块大小、传输的内存位置，而通道方式中这些信息是由通道控制的。另外，每个DMA控制器对应一台设备与内存传递数据，而一个通道可以控制多台设备与内存的数据交换。

5.2、IO核心子系统

1、IO层次结构

IO实现普遍采用了层次式的结构。其基本思想与计算机网络中的层次结构相同：将系统IO的功能组织成一系列的层次，每一层完成整个系统功能的一个子集，其实现依赖于下层完成更原始的功能，并屏蔽这些功能的实现细节，从而为上层提供各种服务。

一个比较合理的层次划分为四个层次的系统结构，各层次及其功能如下：

1）用户层IO软件：实现与用户交互的接口，用户可直接调用在用户层提供的、与IO操作有关的库函数，对设备进行操作。

2）设备独立性软件：用于实现用户程序与设备驱动器的统一接口、设备命令、设备保护，以及设备分配与释放等，同时为设备管理和数据传送提供必要的存储空间。

3）设备驱动程序：与硬件直接相关，用于具体实现系统对设备发出的操作指令，驱动IO设备工作的驱动程序。

4）中断处理程序：用于保存被中断进程的CPU环境，转入相应的中断处理程序进行处理，处理完并回复被中断进程的现场后，返回到被中断进程。

2、IO调度概念

调度一组IO请求就是确定确定一个好的顺序来执行这些请求。应用程序所发布的系统调用的顺序不一定总是最佳选择，所以需要调度来改善系统整体性能，是进程之间公平的共享设备访问，减少IO完成所需要的平均等待时间。

操作系统开发人员通过为每个设备维护一个请求队列来实现调度。当一个应用程序执行阻塞IO系统调用时，该请求就加到相应设备的队列上。IO调度会重新安排队列顺序以改善系统总体效率和应用程序的平均响应时间。

IO子系统还可以使用主存或磁盘上的存储空间的技术，如缓冲、高速缓冲、假脱机等。

3、高速缓存与缓冲区

操作系统总是用磁盘高速缓存技术来提高磁盘的IO速度，对高速缓存复制的访问要比原始数据访问更为高效。例如，正在运行的进程的指令即存储在磁盘上，也存储在物理内存上，也被复制到CPU的二级和一级高速缓存中。

不过，磁盘高速缓存技术不同于通常意义下的介于CPU与内存之间的小容量高速存储器，而是利用内存中的存储空间来暂存从磁盘中读出的一系列盘块中的信息，因此，磁盘高速缓存在逻辑上属于磁盘，物理上则是驻留在内存中的盘块。

高速缓存在内存中分为两种形式：一种是在内存中开辟一个单独的存储空间作为磁盘高速缓存，大小固定；另一种是把未利用的内存空间作为一个缓冲池，共请求分页系统和磁盘IO时共享。

在设备管理子系统中，引入缓冲区的目的有：

1）缓和CPU与IO 设备间速度不匹配的矛盾。

2）减少对CPU的中断频率，放宽对CPU 中断响应时间的限制。

3）解决基本数据单元大小不匹配的问题。

4）提高CPU和IO设备之间的并行性。

其实现方法有：

1）采用硬件缓冲器，但由于成本太高，出一些关键部位外，一般情况下不采用硬件缓冲器。

2）采用缓冲区（位于内存区域）

根据系统设置缓冲器的个数，缓冲技术可以分为：

1）单缓冲。在设备和处理器之间设置一个缓冲区。设备和处理器交换数据时，先把被交换数据写入缓冲区，然后把需要数据的设备或处理器从缓冲区取走数据。

在块设备输入时，假定从磁盘把一块数据输入到缓冲区的时间为T，操作系统将该缓冲区中的数据局传送到用户区的时间为M，而CPU对这一块数据处理的时间为C。由于T和C是可以并行的，所以可把系统对每一块数据的处理时间表示为Max（C,T）+M。

2）双缓冲。双缓冲区机制又称缓冲对换。IO设备输入数据时先输入到缓冲区1，直到缓冲区1满后才输入到缓冲区2，此时操作系统可以从缓冲区1中取出数据放入用户进程，并由CPU计算。双缓冲的使用提高了处理器和输入设备的并行操作的程度。

系统处理一块数据的时间可以粗略地认为是Max（C,T）。如果CT，则可使CPU不必等待设备输入。对于字符设备，若采用行输入方式，则采用双缓冲可使用户再输入完第一行之后，在CPU执行第一行中的命令的同事，用户可继续向第二缓冲区输入下一行数据。而单缓冲情况下则必须等待一行数据被提取完毕才可输入下一行的数据。

如果两台机器之间通信仅配置了单缓冲，那么，他们在任意时刻都只能实现单方向的数据传输。为了实现双向数据传输，必须在两台机器中都设置两个缓冲区，一个用作发送缓冲区，另一个用作接收缓冲区。

3）循环缓冲：包含多个大小相等的缓冲区，每个缓冲区中有一个缓冲区，最后一个缓冲区指针指向第一个缓冲区，多个缓冲区构成一个环形。用于输入输出时，还需要有两个指针in和out。对输入而言，首先要从设备接收数据到缓冲区中，in指针指向可以输入数据的第一个空缓冲区；当运行进程需要数据时，从循环缓冲去中去一个装满数据的缓冲区，并从此缓冲区中提取数据，out指针指向可以提取数据的第一个满缓冲区。输出正好相反。

4）缓冲池：由多个系统共用的缓冲区组成，缓冲区按其使用状况可以形成三个队列：空缓冲队列、装满输入数据的缓冲队列（输入队列）和装满输出数据的缓冲队列（输出队列）。还应具有四种缓冲区：用于收容输入数据的工作缓冲区、用于提取输入数据的工作缓冲区、用于收容输出数据的工作缓冲区、用于提取输出数据的工作缓冲区。

（4）高速缓存与缓冲区的对比

高速缓存是可以保存复制数据的高速存储器。访问高速缓存比访问原始数据更高效，速度更快。

4、设备的分配与回收

设备分配的基本任务是根据用户的IO请求，为他们分配所需的设备。设备分配的总原则是充分发挥设备的使用效率，尽可能地让设备忙碌，又要避免由于不合理的分配方法造成进程死锁。从设备的特性来看，可以把设备分成独占设备、共享设备和虚拟设备三类。

对于独立设备，讲一个设备分配给某进程后，便有改进成都站，直至该进程完成或释放该设备。对于共享设备，可以同时分配给多个进程使用，但需要对这些进程访问该设备的先后次序进程合理的调度。虚拟设备属于可共享设备，可以将它同时分配给多个进程使用。

设备分配依据的主要数据结构有设备控制表（DCT）、控制器控制表（COCT）、通道控制表（CHCT）和系统设备表（SDT），各数据结构功能如下：

设备控制表：系统为每一个设备配置一张DCT，它用于记录设备的特性以及与IO控制器连接的情况。DCT包括设备标示符、设备类型、设备状态、指向COUCT的指针等。其中，设备队列指针指向等待使用该设备的进程组成的等待队列，控制表指针指向于该设备相连接的设备控制器。

控制器控制表：每个控制器都配有一张COCT，它反应设备控制器的使用状态以及和通道的连接情况等。

通道控制表：每个通道配有一张CHCT。

系统设备表：整个系统只有一张SDT，它记录已连接到系统中的所有物理设备的情况，每个物理设备占一个表目。

由于在多道程序系统中，进程数多于资源数，会引起资源的竞争。因此，要有一套合理的分配原则，主要考虑的因素有：IO设备的固有属性，IO设备的分配算法，设备分配的安全性，以及设备独立性。

1）设备分配原则。设备的分配原则应根据设备特性、用户要求和系统配置的情况来决定。设备分配的总原则既要充分发挥设备的使用效率，又要避免造成进程死锁，还要将用户程序和具体设备隔离开。

2）设备分配方式。设备分配方式有静态分配和动态分配两种。

静态分配主要用于对独占设备的分配，它在用户作业开始执行前，有系统一次性分配该作业所要求的全部设备、控制器（和通道）。一旦分配后，这些设备、控制器（和通道）就一直为高作业所占用，知道该作业被撤销。静态分配方式不会出现死锁，但设备的使用效率较低。因此，静态分配方式并不符合分配的总原则。

动态分配是在进程执行过程中根据执行需要进行。当进程需要设备时，通过系统调用命令向系统提出设备请求，由系统按照事先规定的策略给进程分配所需要的设备、IO控制器，一旦用完之后，便立即释放。动态分配方式有利于提高设备的利用率，但如果分配算法使用不当，则有可能造成进程死锁。

3）设备分配算法。常用的动态设备分配算法有先请求先分配、优先级高者优先等。

对于独占设备，即可以采用动态分配方式也可以静态分配方式，往往采用静态分配方式，即在作业执行前，将作业所要用到的这一类设备分配给它。共享设备可被多个进程所共享，一般采用动态分配方式，但在每个IO传输的单位时间内只被一个进程所占有，通常采用先请求先分配和优先级高者先分的分配算法。

设备分配的安全性是指设备分配中应防止发生进程死锁。

1）安全分配方式。每当进程发出IO请求后便进入阻塞状态，直到其IO操作完成时才被唤醒。这样，一旦进程已经获得某种设备后便阻塞，不嫩再请求任何资源，而且在它阻塞时也不保持任何资源。有点事设备分配安全；缺点是CPU和IO设备是串行工作的。

2）不安全分配方式。进程在发出IO请求后继续运行，需要时发出第二个、第三个IO请求等。仅当进程所请求的设备已被另一进程占用时，才进入阻塞状态。有点事一个进程可以同时操作几个设备，从而市金城推进迅速；缺点是这种设备分配有可能产生死锁。

为了提高设备分配的灵活性和设备的利用率、方便实现IO重定向，因此引入了设备独立性。设备独立性是指应用程序独立于具体使用的物理设备。

为了实现设备独立性，在应用程序中使用逻辑设备名来请求使用某类设备，在系统中设置一张逻辑设备表（LUT），用于将逻辑设备名映射为物理设备名。LUT表项包括逻辑设备名、物理设备名和设备驱动程序入口地址；当进程用逻辑设备名来请求分配设备时，系统为他分配相应的物理设备，并在LUT中建立一个表项，以后进程再利用逻辑设备名请求IO操作时，系统通过查找LUT来寻找相应的物理设备和驱动程序。

在系统中可采取两种方式建立逻辑设备表：

1）在整个系统中只设置一张LUT表。这样，所有进程的设备分配情况都记录在这张表中，故不允许有相同的逻辑设备名，主要适用于单用户系统中。

2）为每个用户设置一张LUT。当用户登录时，系统便为用户建立一个进程，同时也位置建立一张LUT，并肩改变放入进程的PCB中。

5、SPOOLing（假脱机技术）

为了缓和CPU的高速型与IO设备低速性之间的矛盾而引入了脱机输入、脱机输出技术。该技术是利用专门的外围控制机，将低速IO设备上的数据传送到高速磁盘上；或者相反。SPOOLing的意思是外部设备同时联机操作，又称为假脱机输入输出操作，是操作系统中采用的一项将独占设备改造成共享设备的技术。

再次攀上开辟出的两个存储区域。输入井模拟脱机输入时的磁盘，用于收容IO设备输入的数据。输出井模拟脱机输出的磁盘，用于收容用户程序的输出数据。

在内存中开辟的两个缓冲区。出入缓冲区用于暂存由输入设备送来的数据，以后再传送到输入井。输出缓冲区用于暂存从输出井送来的数据，以后再传送到输出设备。

输入进程模拟脱机输入时的外围控制机，将用户要求的数据从输入机通过输入缓冲区再送到输入井。当CPU需要输入数据时，直接将数据从输入井读入内存。输入进程模拟脱机输出时的外围控制机，把用户要求输出的数据先从内存送到输出井，待输出设备空闲时，再将输出井中的数据经过输出缓冲区送到输出设备。

共享打印机是使用SPOOLing技术的一个实例，这项技术已被广泛的用于多用户系统和局域网络中。当用户进程请求打印输出时，SPOOLing系统同意为它打印输出，但并不眞正立即把打印机分配给该用户进程，而只为她做两件事：

1）由输出进程在输出井中为之申请一个空闲磁盘块区，并将要打印的数据送入其中。

2）输出进程再为用户进程申请一张空白的用户请求打印表，并将用户的打印要求填入其中，再将该表挂到请求打印队列中。

SPOOLing系统的特点是：提高了IO速度；将独占设备改造为共享设备；实现了虚拟设备功能。

6、出错处理

操作系统可以采用内存保护，这样一来就可以预防许多 硬件和应用程序的错误，即便有一些设备硬件上的适龄也不回导致系统的完全崩溃。

IO设备传输中出现的错误很多，如网络上的堵塞和传输过载等。操作系统可以对一些短暂的出错进行处理，比如读取磁盘出错，那么可以选择重新常识对磁盘进行read操作；再比如在网络上发送数据出错，那么只要网络通信协议允许，就可以做resend操作。但是，如果计算机系统中的重要组件出现了永久性错误，那么操作系统将无法恢复。

作为一个规则，IO系统调用通常返回一位调用状态信息，以表示成功或失败。在UNIX系统中，用一个名为errno的全局变量来表示出错代码，以表示出错原因。

注意：read、send和resend都是操作系统的基本输入输出命令，分别用来读、发送和重发数据。

5.3、本章疑难点

1）分配设备。首先根据IO请求中的物理设备名查找系统设备表（SDT），从中找出该设备的DCT，再根据DCT中的设备状态字段，可知该设备是否正忙。若忙，便将请求IO进程的PCB挂在设备队列上；空闲则按照一定算法计算设备分配的安全性，安全则将设备分配给请求进程，否则仍将其PCB挂到设备队列。

2）分配控制器。系统把设备分配给请求IO的进程后，再到其DCT中找出与该设备连接的控制器的COCT，从COCT中的状态字段中可知该控制器是否忙碌。若忙，便将请求IO进程的PCB挂在该控制器的等待队列上；空闲便将控制器分配给进程。

3）分配通道。在该COCT中又可找到与该控制器连接的通道CHCT，再根据CHCT内的状态信息，可知该通道是否忙碌。若忙，便将请求IO的进程挂在该通道的等待队列上；空闲便将该通道分配给进程。只有在上述三者都分配成功时，这次设备分配才算成功。然后，便可启动该IO设备进行数据传送。

为使独占设备的分配具有更强的灵活性，提高分配的成功率，还可以从以下两方面对基本的设备分配程序加以改进：

1）增加设备的独立性。进程使用逻辑设备名请求IO。这样，系统首先从SDT中找出第一个该类设备的DCT。若该设备忙，又查找出第二个该设备的DCT。仅当所有该类设备都忙时，才把进程挂在该类设备的等待队列上；只要有一个该类设备可用，系统便进一步计算分配该设备的安全性。

2）考虑多通路情况。为防止IO系统的“瓶颈”现象，通常采用多通路的IO系统结构。此时对控制器和通道的分配同样要经过几次反复，即若设备（控制器）所连接的第一个控制器（通道）忙时，应查看其所连接的第二个控制器（通道），仅当所有的控制器（通道）都忙时，此次的控制器（通道）分配才算失败，才把进程挂在控制器（通道）的等待队列上。而只要有一个控制器（通道）可用，系统便可将它分配给进程。